НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
“___”_____20__ р

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп’ютерна інженерія»

на тему: «Модуль обчислення квадратного кореня з плаваючою комою»

Виконав:
студент _IV_курсу, групи__ІО-62__

____Трофимчук Дмитро Андрійович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _доц. д.т.н Сергієнко Анатолій Михайлович_ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант __н. контроль_ _доц. д.т.н. Сімоненко В.П._ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент _доц. каф. СПіСКС к.т.н.,доц Орлова М.М__ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Сергій СТІРЕНКО*
(підпис)
"___"_____20__ р.

ЗАВДАННЯ

на дипломний проект студенту

Трофимчуку Дмитру Андрійовичу

1. Тема проекту «*Модуль обчислення квадратного кореня з плаваючою комою»*

Керівник дипломного проекту: доц., д.т.н. Сергієнко Анатолій Михайлович, затверджені наказом по університету від «_____» _____ 2020 року № ____

2. Термін здачі студентом закінченої роботи_____*2020р.*

3. Вихідні дані до проекту *технічне завдання, теоретичні дані.*

4. Зміст пояснювальної записки: *огляд існуючих ПЛІС, програмних та апаратних середовищ для розробки, алгоритмів, висновки..*

5. Консультант роботи, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В.П.		

6. Дата видачі завдання *01.09 .2019 року*

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1.	*Затвердження теми роботи*	01.09.2019	
2.	*Вивчення та аналіз завдання*	17.09.2019	
3.	*Розробка архітектури та загальної структури систем*	04.10.2019	
4.	*Розробка структур окремих підсистем*	18.12.2019	
5.	*Програмна реалізація системи*	03.02.2020	
6.	*Оформлення пояснювальної записки*	04.04.2020	
7.	*Передзахист*	05.05.2020	
8.	*Захист*	19.06.2020	

Студент Трофимчук Дмитро _____

(підпис)

Керівник Анатолій Сергієнко _____

(підпис)

Анотація

Робота присвячена розробці Модуль обчислення квадратного кореня з плаваючою комою на базі ПЛІС. Через зростання актуальності розрахунків з плаваючою комою для графіки, нейронних мереж, безпілотних автомобілів, та інших приладів. Запропонованій модуль являється математичним співпроцесором для розрахунку чисел з плаваючою комою. Це дозволить скоротити час на розрахунки, та збільшити їх точність.

Abstract

The work is devoted to the development of the module of calculation of a square root with a floating point on the basis of FPGA. Due to the growing relevance of floating point calculations for graphics, neural networks, unmanned vehicles, and other devices. The proposed module is a mathematical coprocessor for calculating floating point numbers. This will reduce the time for calculations and increase their accuracy.

	Формат	Позначення	Найменування	Листів	№ екзем.	Примітка
			Документація загальна			
	А4	ІАЛЦ.467450.002 ТЗ	Модуль обчислення	4		
			квадратного кореня з			
			плаваючою комою			
	А4	ІАЛЦ.467450.003 ПЗ	Модуль обчислення	50		
			квадратного кореня			
			Пояснювальна записка			
	А4	ІАЛЦ.467450.004 А1	Модуль обчислення	18		
			квадратного кореня			
			Лістинг програми			
	А4	ІАЛЦ.467450.005 Д1	Модуль обчислення	1		
			квадратного кореня			
			Принципова схема			
	А4	ІАЛЦ.467450.006 Д2	Модуль обчислення	1		
			квадратного кореня			
			Функціональна схема			
	А4	ІАЛЦ.467450.007 Д3	Модуль обчислення	1		
			квадратного кореня			
			Структурна схема			
		CD-RW		1		

					ІАЛЦ.467450.001 ВП			
		ПІБ	Підпис	Дата				
Розроб.		Трофимчук Д,А.			Модуль обчислення квадратного кореня з плаваючою комою Відомість дипломного проекту	Літ.	Лист	Листів
Перевір.		Сергієнко А.М.					1	1
						НТУУ «КПІ», ФІОТ ІО-62		
Н. контр.		Сімоненко В. П.						
Зав. каф.		Стіренко С. Г.						

Технічне завдання

до дипломного проекту

на тему: «Модуль обчислення квадратного кореня з плаваючою комою»

Київ – 2020

ЗМІСТ

1. **НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ**

Технічне завдання розповсюджується на розробку апаратного та програмного продукту з теми «Модуль обчислення квадратного кореня з плаваючою комою». Область застосування – комп'ютерні системи.

2. **ПІДСТАВИ ДЛЯ РОЗРОБКИ**

Підставою для розробки є технічне завдання на виконання бакалаврського дипломного проекту, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київського політехнічного інституту імені Ігоря Сікорського».

3. **МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ**

Метою розробки є проектування модулю обчислення квадратного кореня з плаваючою комою, який розрахований на модифікацію існуючих систем.

Призначенням розробки є збільшення швидкодії та обробки розрахунків існуючих систем.

4. **ДЖЕРЕЛА РОЗРОБКИ**

Джерелом розробки є науково-технічна література з розробки ПЛІС, побудови апаратних та програмних продуктів, та публікації в Інтернеті з даної теми.

5. **ТЕХНІЧНІ ВИМОГИ**

5.1. **Вимоги до програмного продукту**

Сирцевий код інтерпретованої мови програмування повинен мати розширення «.air», яке повинно бути зареєстрованим в середовищі розробки IntelliJ IDEA Ultimate Edition.

Інтерпретована мова програмування має бути інтегрована в середовище розробки IntelliJ IDEA Ultimate Edition для подальшого рефакторингу сирцевого коду та відлагодження програмного забезпечення.

5.2. **Вимоги до програмного забезпечення**

Для запуску в середовища розробки Xilinx Vivado потрібно мати операційну систему Windows (версії 8/10 x64).

5.3. **Вимоги до апаратної частини**

Для запуска середовища розробки Xilinx Vivado потрібно мати вільне місце на жорсткому диску не менше 25 Гб, екран з мінімальним розширенням 1024x768, оперативну пам’ять не менше 4 Гб.

6. **ЕТАПИ РОЗРОБКИ**

Розробка програмного продукту у вигляді інтерпретованої мови програмування складається з наступних етапів:

1) затвердження теми роботи;

2) вивчення та аналіз завдання;

3) вивчення літературних джерел;

4) розробка програмного продукту;

5) оформлення пояснювальної записки;

6) попередній захист дипломного проекту;

7) захист дипломного проекту.

Пояснювальна записка

до дипломного проекту

на тему: «Модуль обчислення квадратного кореня з плаваючою комою»

Київ – 2020

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

ПЛІС – програмована логічна інтегральна схема— електронний компонент, що використовується для створення цифрових інтегральних схем. На відміну від звичайних цифрових мікросхем, логіка роботи ПЛІС не визначається при виготовленні, а задається за допомогою програмування.

САПР - система автоматизованого проектування,автоматизована система, що реалізує інформаційну технологію виконання функцій проектування.

ПЛМ - програмовані логічні матриці.

ПМЛ - програмована матрична логіка.

ПМ - програмована макрологіка.

ПКМБ - програмовані комутовані матричні блоки.

ПЗМ - програмовані вентильні матриці

Verilog - (англ. Verilog Hardware Description Language) це мова опису апаратури, що використовується для опису і моделювання електронних систем.

VHDL - (англ. VHSIC (Very high speed integrated circuits) Hardware Description Language) - мова опису апаратури інтегральних схем. Мова проектування VHDL є базовою мовою при розробці апаратури сучасних обчислювальних систем.

Vivado Design Suite - це програмний пакет, що виробляється Xilinx для синтезу та аналізу HDL-конструкцій.

ВСТУП

На сьогоднішній день обчислення з плаваючою комою використовується повсюдно. Зазвичай використовуються процесори які обладнані більш на цілі числа, ніж на числа з плаваючою комою. Це накладає певні обмеження на швидкість і точність розрахунків. Для обходу цих обмежень використовують математичні співпроцесори або спеціально підготовлені плати на ПЛІС. Також ПЛІС застосовуються, як прискорювачі універсальних процесорів в суперкомп'ютерах.

Актуальність теми

У нинішній час популярність набирають нейронні мережі, безпілотні дрони і автомобілі на них. Нейронні мережі як і GPS використовують числа з плаваючою комою, тому модулі для прорахунку цих чисел можуть істотно збільшити швидкість і точність роботи, а також за рахунок підвищеної точності розрахунків, убезпечити пристрої та людей від непередбачених помилок.

Мета і задачі дослідження

Метою роботи є розробка модуля обчислення квадратного кореня з плаваючою комою. Що дозволить скоротити сумарний час обчислень.

Для досягнення поставленої мети були поставлені наступні основні задачі:

- Провести аналіз мікоросхем ПЛІС та математичних співпроцесорів, що виконують схожі задачі;
- Створити модель та програмну реалізацію розробленого методу;
- Провести моделювання запропонованого методу;
- Дослідити та проаналізувати отримані характеристики підчас моделювання.

РОЗДІЛ 1.
ОГЛЯД МІКРОСХЕМ ПЛІС

1.1. Класифікація ПЛІС

Програмовані логічні інтегральні схеми або ПЛІС – це напівпровідникові пристрої, що базуються на матриці логічних блоків, які можливо конфігурувати через програмовані зв'язки. Після виготовлення, ПЛІС можуть бути сконфігуровані для виконання необхідних функцій. Ця особливість відрізняє їх від спеціалізованих інтегральних схем, які виготовляються під конкретну задачу.

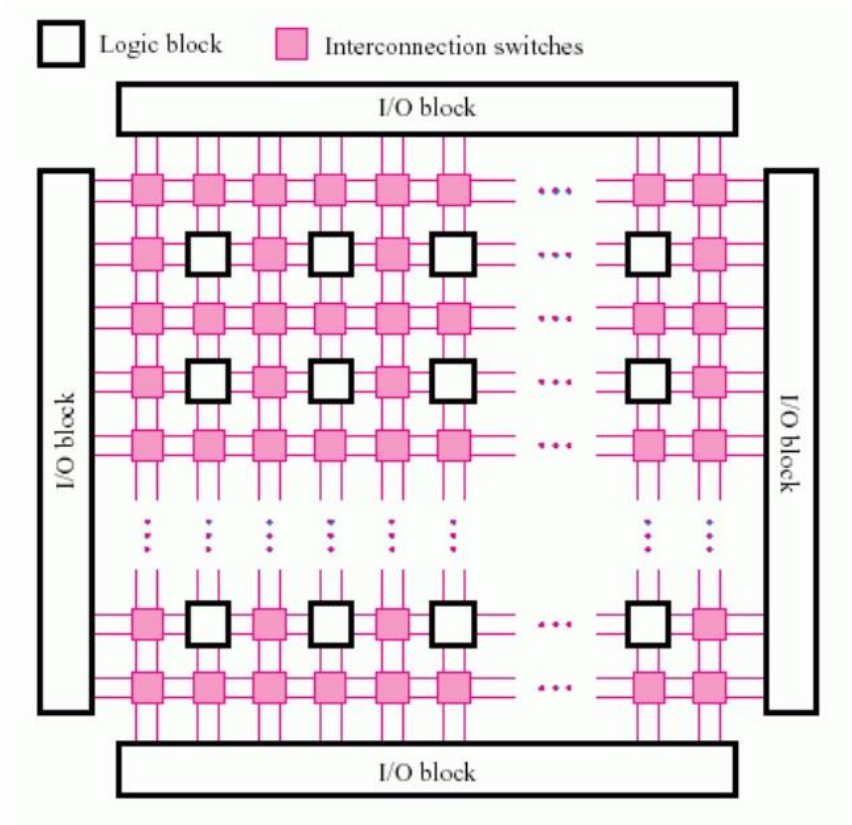


Рис.1.1 Узагальненена структура ПЛІС

Здебільшого програмовані логічні інтегральні схеми складаються із наступних елементів:

- Конфігуруванні логічні блоки, що реалізують певну логічну функцію;
- Програмованих електронних зв’язків між конфігурованими логічними блоками;
- Програмованих блоків вводу/виводу, що забезпечують зовнішній зв’язок із внутрішньою логікою мікросхеми.

Найбільший інтерес представляє класифікація ПЛІС по структурам, тому що вона дає найбільш повне уявлення про клас задач. Основним критерієм такої класифікації є наявність, вид і способи комутації логічних матриць. зацією ознакою можна виділити наступні класи ПЛІС:

- програмовані логічні матриці (ПЛМ);
- програмована матрична логіка (ПМЛ);
- програмована макрологіка (ПМ);
- програмовані комутовані матричні блоки (ПКМБ);

програмовані вентильні матриці (ПЗМ)

1.2. Програмовані логічні матриці

Програмовані логічні матриці найбільш традиційний тип ПЛІС, має програмовані матриці «І» і «АБО». У зарубіжній літературі відповідними цього класу абревіатурами є FPLA (Field Programmable Logic Array) і FPLS (Field Programmable Logic Sequensers).

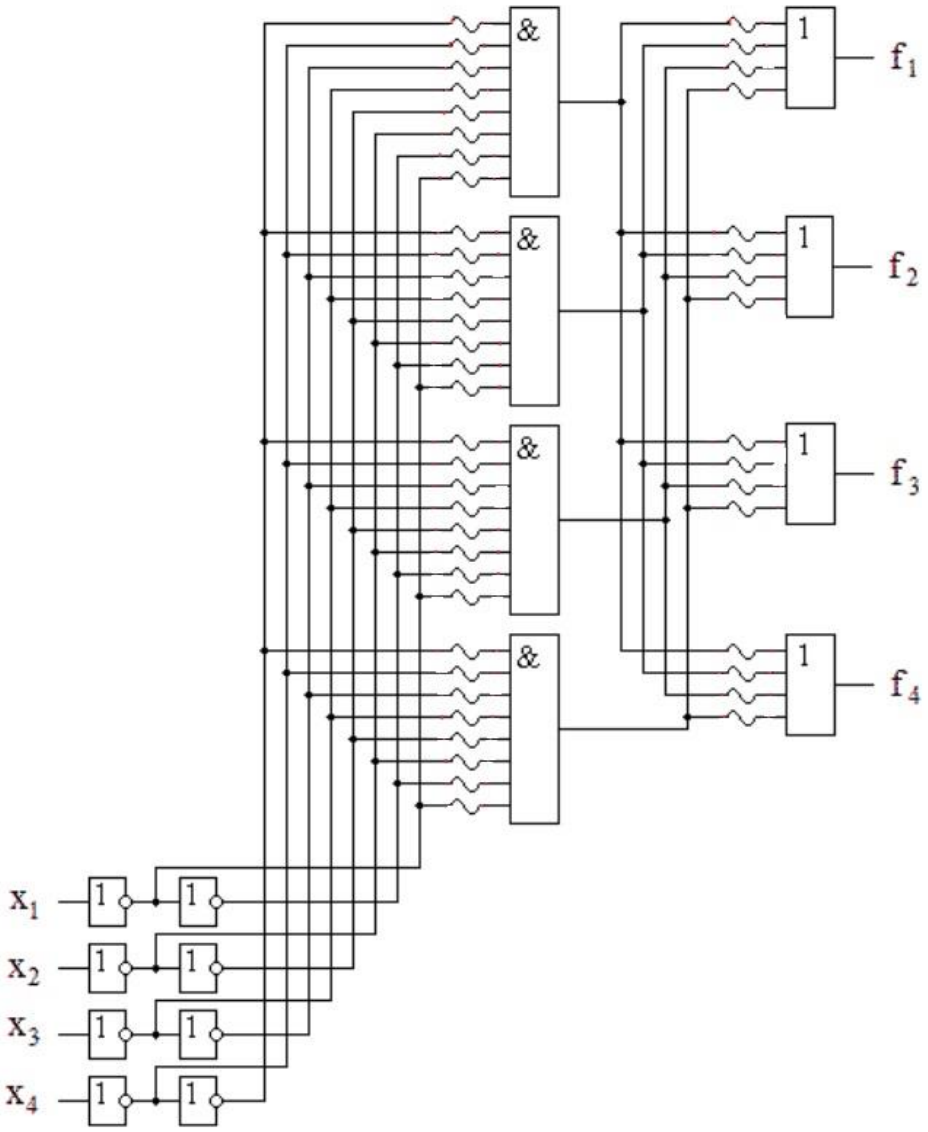


Рис.1.2 Приклад ПЛМ

Недоліком структури ПЛМ є слабке використання ресурсів програмованої матриці АБО. Тому була запропонована більш проста, але тим, більш ефективна архітектура програмованої матричної логіки (ПМЛ). В англійській термінології -Programmable Array Logic (PAL).

1.3. Програмована матрична логіка

Програмована матрична логіка - це ПЛІС, що мають програмовану матрицю «І» та фіксовану матрицю «АБО». До цього класу належать більшість сучасних ПЛІС. Як приклади можна привести ІС КМ1556ХП4, ХП6, ХП8, ХЛ8, ПЛІС фірм INTEL, ALTERA, AMD, LATTICE і ін. Різновидом класу ПМЛ є ПЛІС, що мають тільки одну (Програмовану) матрицю «І», наприклад, схема 85С508 фірми INTEL.

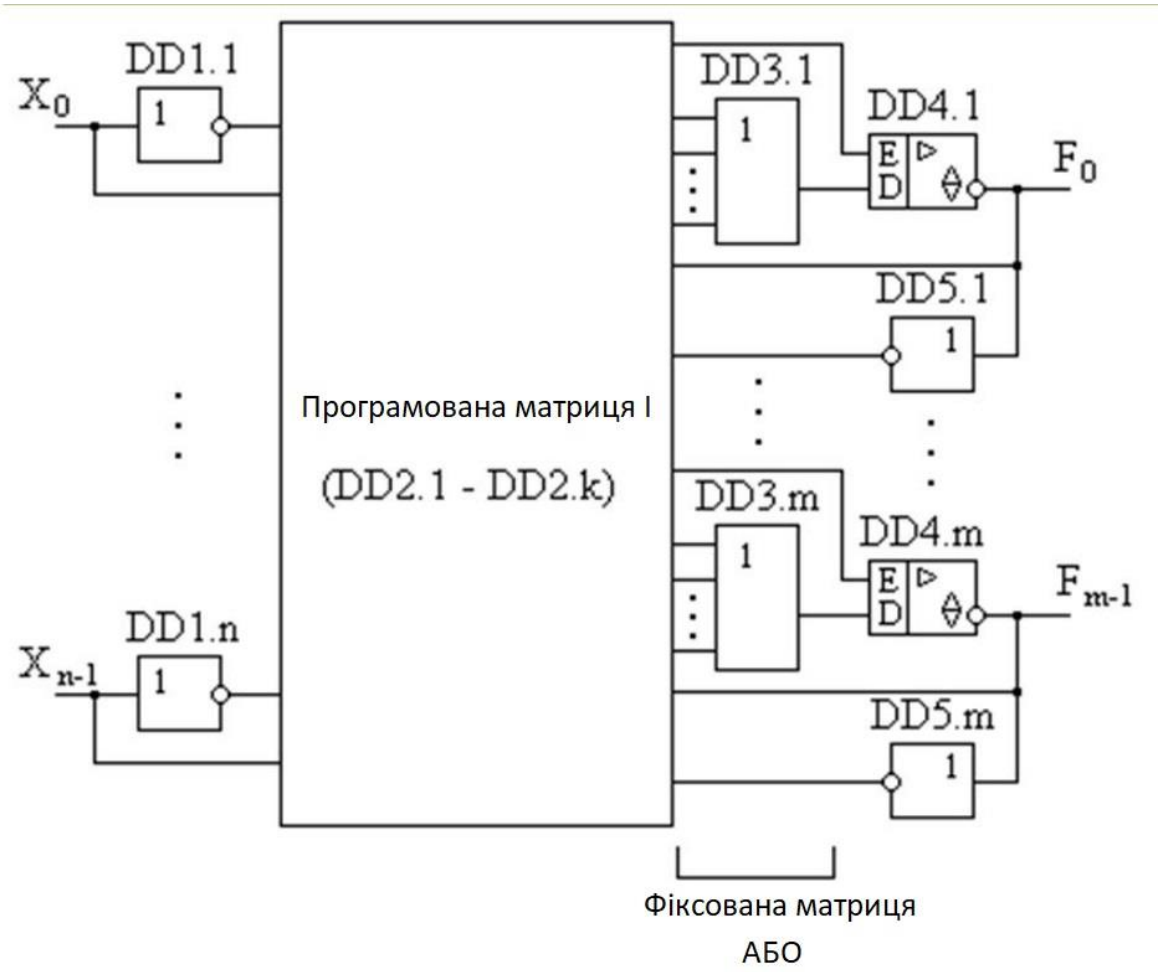


Рис.1.3 Приклад ПМЛ

З малюнка видно, що входи елементів АБО DD3 не є комутованими. Таким чином, на кожен вхід поточного елемента АБО подається певний вихід елемента І програмованої матриці І.

1.4. Програмована макро логіка

Програмована макрологіка - ПЛІС, що містять одну програмовану матрицю «І-НЕ» або «АБО-НЕ», але за рахунок численних інверсних зворотних зв'язків здатні формувати складні логічні функції. До цього класу належать, наприклад, ПЛІС PLHS501 і PLHS502 фірми SIGNETICS, що мають матрицю «І-НЕ», а також схема XL78C800 фірми EXEL, заснована на матриці «АБО-НЕ»

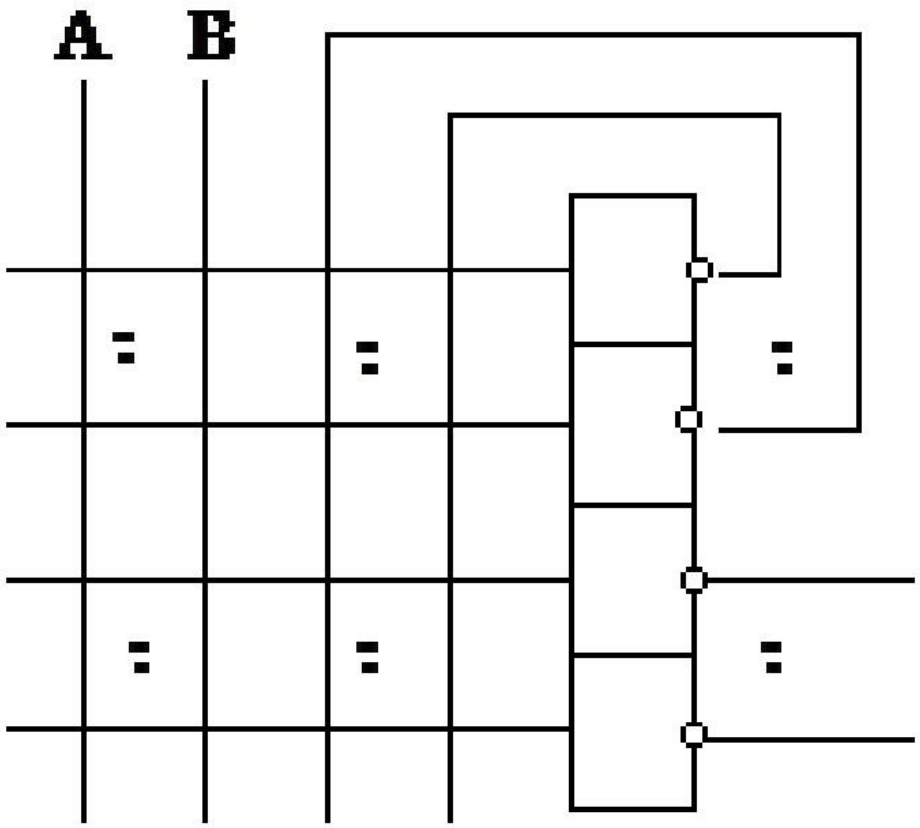


Рис.1.4 Приклад програмованої макрологіки

Елементів може бути досить багато, наприклад, у сучасних ПЛІС ємність до 1 млн.

1.5. Програмовані комутовані матричні блоки

Програмовані комутовані матричні блоки - це ПЛІС, що містять кілька (зазвичай 4-8) матричних логічних блоків (МЛБ), об'єднаних комутаційної матрицею. Кожен МЛБ є структурою типу ПМЛ, тобто програмовану матрицю «І», фіксовану матрицю «АБО» і макроячейки. ПЛІС типу ПКМБ, як правило, мають високу ступінь інтеграції (до 20000 вентилів).

До цього класу належать ПЛІС сімейства MAX5000, MAX7000, і MAX9000 фірми ALTERA, схеми сімейств XC72xx, XC73xx і XC95xx фірми XILINX.

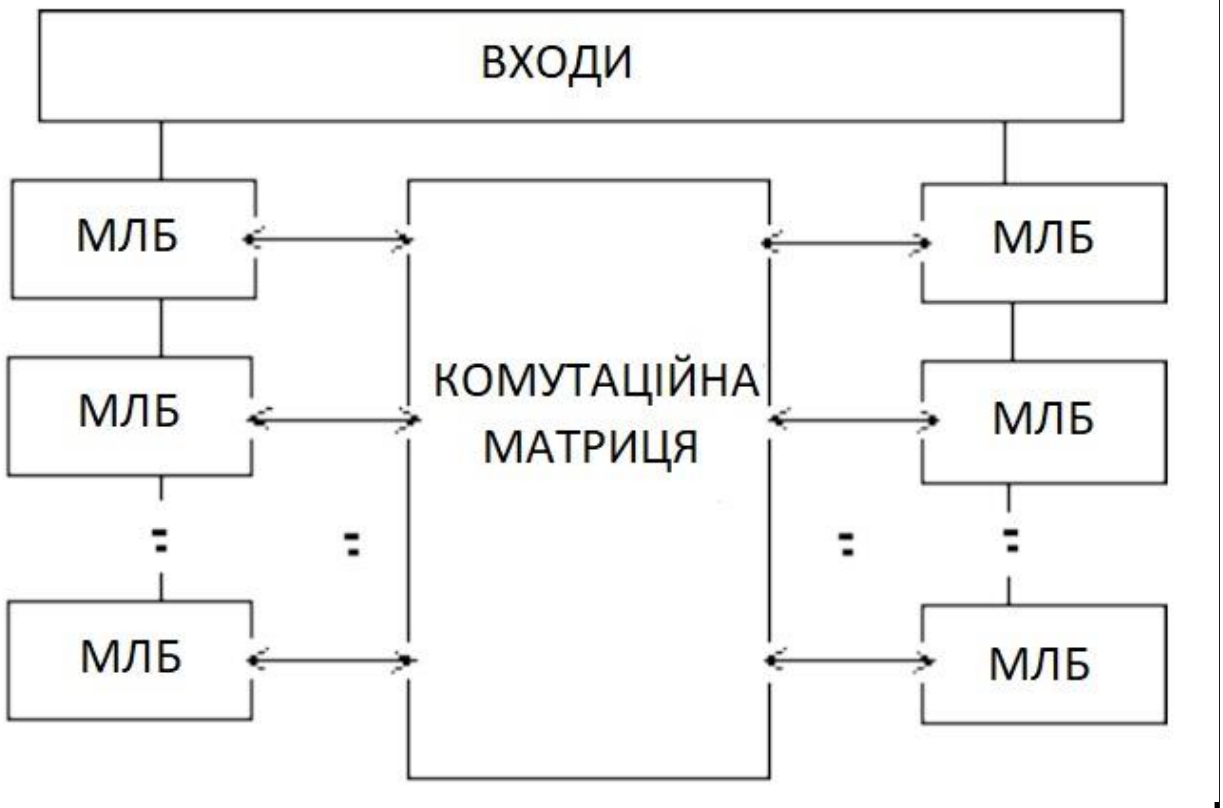


Рис.1.5 Приклад програмованхи комутативних матричних блоків

1.6. Програмовані вентильні матриці

Програмовані вентильні матриці (ПЗМ), складаються з логічних блоків (ЛБ) і комутуючих шляхів - програмованих матриць з'єднань. Логічні блоки таких ПЛІС складаються з одного або декількох ві дносно простих логічних елементів, в основі яких лежить таблиця перекодування (ТП, Look-up table - LUT), програмований мультиплексор, D-тригер, а також кола керування.

Застосування: ПВМ використовуються, головним чином, для розробки рахункових структур з мінімальним комбінаційною обрамленням. До ПВМ класу відносяться схеми фірм XILINX, ACTEL, а також сімейства FLEX8000, FLEX10K і FLEX6000 фірми ALTERA.

Саме цей тип ПЛІС програмується за допомогою таких мов програмування як Verilog та VHDL.

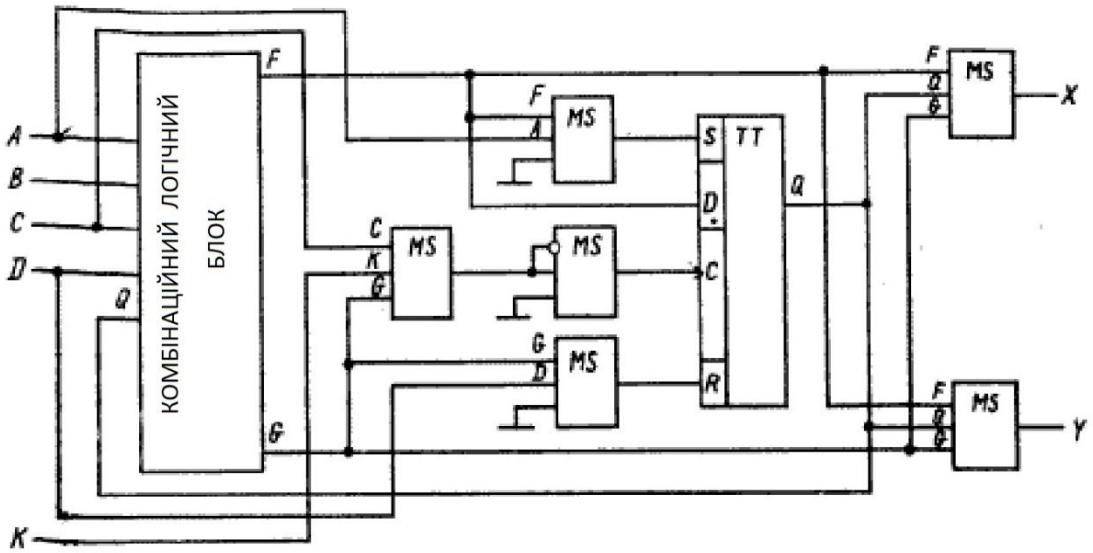


Рис1.6 Приклад програмованої вентильної матриці

Сучасні сімейства ПВМ включають вбудовану функціональність високого рівня.

ВИСНОВКИ ДО РОЗДІЛУ 1

Існують різні ПЛІС, як по виробникам так і за своєю структурою, яка в подальшому визначає застосування тієї чи іншої мікросхеми, але в сучасних реаліях найпопулярнішими являються схеми з програмно вентельний матрицями.

Однією з причин їх розповсюдження являються мови програмування Verilog та VHDL, а також найбільша здатність до перепрограмування та зміни топології зв'язків у процесі користування.

РОЗДІЛ 2.

Аналіз програмних та апаратних середовищ для розробки

2.1. Мови опису апаратури

Для розробки на ПЛІС використовують мови опису апаратури, а також середовища для розробки на подобі Quartus II від Altera.

Як правило в таких середовищах проходить весь процес розробки і тестування пристрою. Вони володіють гнучким інтерфейсом, вбудованими емуляторамі і компіляторами, а так само засобами налагодження.

Мова опису апаратури — це спеціалізована формальна комп'ютерна мова, що використовується для проектування структури, дизайну та роботи електронної мікросхеми та її моделювання. Вона дає можливість автоматично аналізувати, імітувати та тестувати створюваний пристрій. Компілятор мусить забезпечувати переведення програми, написаної на будь-якій з мов опису апаратури на низькорівневу специфікацію фізичних електронних компонентів з ціллю створити мікросхему.

До переваг використання мови проектування апаратури над схемним проектуванням можна віднести наступні:

- можливість проектування пристроїв, для яких створення схем неможливе через надмірну складність продукту, наприклад при розробці сучасних процесорів;
- такий опис зазвичай об'єднує як структурну, так і функціональну складову;
- проект на мові проектування апаратури портативний та універсальний, тобто легко переноситься на іншу елементну базу і може використовуватися в іншому проекті. Як результат, він стає довгоживучим, оскільки не залежить від «старіння» конкретних мікросхем;

- мови опису апаратури гарантують високу надійність розроблюваного пристрою та забезпечують самодокументування.

Мови проектування апаратури дають можливість інженерам працювати на вищому рівні абстракції, що дозволяє збільшити кількість елементів розроблюваного пристрою.

Мови опису апаратури класифікують за їх цільовим призначенням. До першої категорії відносять ті, що дозволяють виконувати моделювання аналогових та гібридних мікросхем. До другої — ті, що моделюють цифрові мікросхеми.

Табл.2.1 Мови для аналогових схем

Назва	Опис
Analog Hardware Descriptive Language	Мова опису для програмованих аналогових інтегральних схем
SpectreHDL	Мова проектування аналогової апаратури
Verilog-AMS (англ. *Verilog for Analog and Mixed-Signal*)	Стандартизоване розширення Verilog для аналогового та гібридного моделювання
VHDL-AMS (англ. *VHDL with Analog/Mixed-Signal extension*)	Стандартизована мова для гібридного моделювання
HDL-A	Мова опису аналогового апаратного забезпечення

На даний момент мови для аналогових схем майже не використовують, через відсутність спросу на аналогові схеми.

Більшою популярністю користуються мови для цифрових схем ці мови представлені у таблиці 2.1.

Табл. 2.2 Мови для цифрових схем

Назва	Опис
AHDL (англ. *Altera Hardware Description Language*)	Мова опису апаратури компанії Altera
Bluespec	Високорівнева мова проектування апаратури, заснована на мові програмування Haskell
BSV (англ. *Bluespec SystemVerilog*)	Мова опису апаратури, заснована на Bluespec з синтаксисом, подібним до Verilog
Chisel (англ. *Constructing Hardware in a Scala Embedded Language*)	Мова проектування апаратури, заснована на мові програмування Scala
HML (англ. *Hardware ML*)	Мова проектування апаратури, заснована на SML[6]
Hydra (мова опису апаратури)	Мова опису апаратури, заснована на Haskell[7]
M (мова опису апаратури)	Мова опису апаратури, розроблена компанією Mentor Graphics
MyHDL	Мова проектування апаратури, заснована на Python
ParC (англ. *Parallel C++*)	C++, оптимізований для задач проектування апаратури
PHDL (англ. *Printed Circuit Board Hardware Description Language*)	Мова опису апаратури з відкритим сирцевим кодом для поєднання друкованих плат
SystemVerilog	Надбудова над Verilog
SystemTCL	Мова опису апаратури, заснована на Tcl
THDL++ (англ. *Templated Hardware Description Language ma C++*)	Розширення VHDL
Verilog	Одна з найбільш поширених мов проектування апаратури
VHDL (англ. *Very High Speed Integrated Circuit Hardware Language*)	Одна з найчастіше використованих мов опису апаратури

2.1. Огляд мови Verilog

Verilog HDL (англ. Verilog Hardware Description Language) — мова опису апаратури (HDL), що використовується для опису та моделювання електронних систем. Verilog HDL не слід плутати з VHDL, найбільш часто використовується у проектуванні, верифікації і реалізації аналогових, цифрових та змішаних електронних систем на різних рівнях абстракції.

Розробники Verilog зробили його синтаксис дуже схожим на синтаксис мови C, що спрощує його освоєння. Verilog має препроцесор, дуже схожий на препроцесор мови C, і основні керуючі конструкції if, while також подібні однойменним конструкціям мови C. Угоди по форматуванню виведення також дуже схожі.

Слід зазначити, що опис апаратури, написаний мовою Verilog (як і іншими HDL-мовами) прийнято називати програмами, але, на відміну від загальноприйнятого поняття програми, як послідовності інструкцій, тут програма представляє множину операторів, які виконуються паралельно і циклічно під керуванням об'єктів, названих сигналами. Кожен такий оператор є моделлю певного елемента реальної функціональної схеми апаратури, а сигнал — аналогом реального логічного сигналу. Так само для мови Verilog не застосовується термін «виконання програми». Фактично, виконання Verilog-програми є моделюванням функціональної схеми, яку вона описує, що виконується спеціальною програмою — Verilog-симулятором.

Розробники мови Verilog хотіли створити її за синтаксисом подібною до мови програмування C, яка уже широко використовувалася при розробці програмного забезпеченні. Як і C, Verilog чутливий до регістру і має базовий препроцесор (хоча не такий складний як у ANSI C/C++). Його ключові слова для керування потоком (такі як if/else, for, while, case, та інші) є еквівалентними, а Черговість операцій сумісна із C. До синтаксичних відмінностей відносяться: необхідність вказувати ширину в бітах при декларації змінних, демаркація процедурних блоків (Verilog використовує

ключові слова begin/end замість фігурних дужок {}), і багато інших не значних відмінностей. Verilog вимагає, щоб усім змінним визначався розмір. В C ці розміри визначаються 'типом' змінної (наприклад, цілий тип може мати розмір в 8 біт).

Структура програми на Verilog складається із ієрархії модулів. Модулі інкапсулюють ієрархію дизайну, і комунікують з іншими модулями через множину оголошених входів, виходів і двонаправлених портів.

Існує підмножина інструкцій мови Verilog, придатна для синтезу. Модулі, які написані в межах цієї підмножини, називають RTL (англ. register transfer level — рівень регістрових передач). Вони можуть бути фізично реалізовані з використанням САПР (Система автоматизованого проектування і розрахунку) синтезу. САПР за певними алгоритмами перетворить абстрактний вихідний Verilog-код на перелік зв'язків— логічно еквівалентний опис, що складається з елементарних логічних примітивів (наприклад, елементи AND, OR, NOT та тригери), які доступні у вибраній ПЛІС.

Verilog містить два базових типу даних: wire та reg. Обидва ці типу можуть приймати 4 можливих значення при симуляції Verilog програми:

- 0
- 1
- X — «Невідоме значення». Це значення використовується тільки для симуляції, в реальній апаратурі буде 0 або 1.
- Z — «Стан високого опору», тобто відсутність сигналу.

Тип wire використовується для опису ланцюгів, reg для регістрів і змінних. Обидва ці типу можуть також бути використані при описі багатобітових даних.

Крім цього, Verilog містить ще наступні типи даних:

- integer - для обчислення поточного часу в процесі моделювання, розмір 32 біт
- real - підтримує використання дійсних чисел, констант і змінних.
- time - для обчислення поточного часу в процесі моделювання, розмір 64біт
- realtime

Verilog містить два види блоків, які можуть виробляти обчислення: «initial» -блок і «always» -блок.«Initial» -блок визначає, які дії повинні бути зроблені при старті програми. Цей блок не є синтезованим і зазвичай використовується для тестування.

Програма може містити кілька «initial» Блоки, всі вони виконуються паралельно.

2.3. Огляд мови VHDL

VHDL (англ. VHSIC (Very high speed integrated circuits) Hardware Description Language) — мова опису апаратури інтегральних схем. Мова проектування VHDL є базовою мовою при розробці апаратури сучасних обчислювальних систем.

Мова VHDL створена як засіб опису цифрових систем, однак існує підмножина мови — VHDL AMS (аналогових та змішаних сигналів), що дозволяє описувати як чисто аналогові, так і змішані, цифро-аналогові схеми.

Проект в системі проектування на основі VHDL представляється сукупністю ієрархічно пов'язаних текстових фрагментів, званих проектними модулями. Розрізняють первинні і вторинні проектні модулі

Приклда :

<Первинний модуль> :: =

<Декларація суті>

| <Декларація пакета>

| <Декларація конфігурації>

<Вторинний модуль> :: =

<Архітектурне тіло>

| <Тіло пакета>

Декларація суті (entity) визначає ім'я проекту і його інтерфейс, тобто порти і параметри налаштування. Архітектурне тіло сутності описує той чи інший спосіб функціонування пристрою і (або) його структуру. Кожній сутності зіставляється одне або кілька архітектурних тіл. Кілька вторинних модулів, що відповідають одному первинному, складають набір можливих альтернативних реалізацій об'єкта, представленого первинним модулем. Наприклад, однієї сутності може відповідати кілька архітектурних тел, що відрізняються ступенем деталізації опису і навіть алгоритмом перетворення даних.

Первинні і відповідні їм вторинні модулі можуть зберігатися в різних файлах або записуватися в одному файлі. Важливо лише, щоб вони були скомпільовані в загальну проектну бібліотеку, причому первинний модуль компілюється раніше підлеглого йому вторинного. При записи первинного і вторинного модулів в одному файлі первинний модуль записується раніше відповідного йому вторинного.

Первинні і відповідні їм вторинні модулі можуть зберігатися в різних файлах або записуватися в одному файлі. Важливо лише, щоб вони були скомпільовані в загальну проектну бібліотеку, причому первинний модуль компілюється раніше підлеглого йому вторинного. При записи первинного і вторинного модулів в одному файлі первинний модуль записується раніше відповідного йому вторинного.

Типовий текст програми на VHDL має наступну структуру:

<VHDL-програма> :: =

{

{<Оголошення бібліотеки>}

{<Оголошення використання>}

<Первинний модуль>

}

{<Вторинний модуль>}

<Оголошення бібліотеки> :: = library <ім'я бібліотеки>;

<Оголошення використання> :: =

use <ім'я бібліотеки>. <назва пакунка>. <ім'я модуля>;

Іншими словами, текст являє собою довільний набір первинних і вторинних модулів, причому кожного первинного модулю може передувати вказівка на бібліотеки і пакети, інформація з яких потрібно для побудови цього модуля. Після цього необхідно вказати, що вони будуть використані за допомогою відповідного оголошення. Як ім'я модуля часто вживають all - в цьому випадку будуть використані всі наявні модулі. Відзначимо, що навіть якщо кілька первинних модулів посилаються на одні і ті ж бібліотеки і модулі, декларація використання повинна передувати кожному первинному модулю окремо.

Синтаксис декларації суті має вигляд:

<Декларація суті> :: =entity <ім'я проекту> is

[<Оголошення параметрів настройки>]

[<Оголошення портів>]

[<Розділ декларацій>]

[Begin <розділ опреатора>]

end [entity] <ім'я проекту>;

<Оголошення параметрів настройки> :: =

generic (<ім'я>: <тип> [: = <вираз>]

{; <Ім'я>: <тип> [: = <вираз>]});

<Оголошення портів> :: =

port (<ім'я>: <режим> <тип> [: = <вираз>]

{; <Ім'я>: <режим> <тип> [: = <вираз>]};

<Режим> :: = in | out | inout

Архітектурні тіла представляють змістовне опис проекту. Архітектурне тіло в певному сенсі підпорядковане відповідної сутності. Розрізняють структурні архітектурні тіла (описують проект у вигляді сукупності компонентів і їх з'єднань), поведінкові архітектурні тіла (описують проект як сукупність виконуваних дій) і змішані тіла. Формальні ознак, за якими тіло можна було б віднести до того чи іншого типу, не існує - мова йде скоріше не про чітку класифікацію, а про стилях опису, для кожного з яких характерними є певні оператори і які краще відображають різні аспекти одного і того ж об'єкта. Визначено наступні правила записи архітектурних тіл:

<Архітектурне тіло> :: =

architecture <ім'я архітектури> of <ім'я сутності> is

<Розділ декларацій>

begin

<Розділ операторів>

end [architecture] <ім'я архітектури>;

<Arkhitekturne tilo> :: =

Тут ім'я архітектури - це будь-який індивідуальне ім'я проектного модуля. ім'я суті задає первинний модуль, якому підпорядковується архітектурне тіло. В розділі декларацій оголошуються локальні для цього модуля інформаційні одиниці – типи даних, сигнали, підпрограми тощо Розділ операторів описує правила функціонування і (або) конструювання пристрою в термінах мови.

Мова VHDL заснована на концепції суворої типізації даних, тобто будь-якій одиниці інформації в програмі обов'язково присвоюється ім'я, і для неї повинен бути визначений тип. Визначення інформаційної одиниці розміщується в розділі декларацій програмного модуля, в якому воно використовується, або ієрархічно попереднього модуля. Тип даних визначає набір значень об'єктів, віднесених до цього типу, а також набір допустимих перетворень цих даних. Дані різних типів несумісні в одному виразі.

Мова VHDL надає певний базовий набір типів даних, які не вимагають оголошення в програмі користувача. Крім того, користувач може визначити свої типи даних і підключити існуючі з бібліотек. Розрізняють скалярні типи даних і агрегатні типи. Об'єкт, віднесений до скалярного типу, розглядається як закінчена одиниця інформації. Агрегат являє упорядковану сукупність скалярних одиниць, об'єднаних єдиним ім'ям.

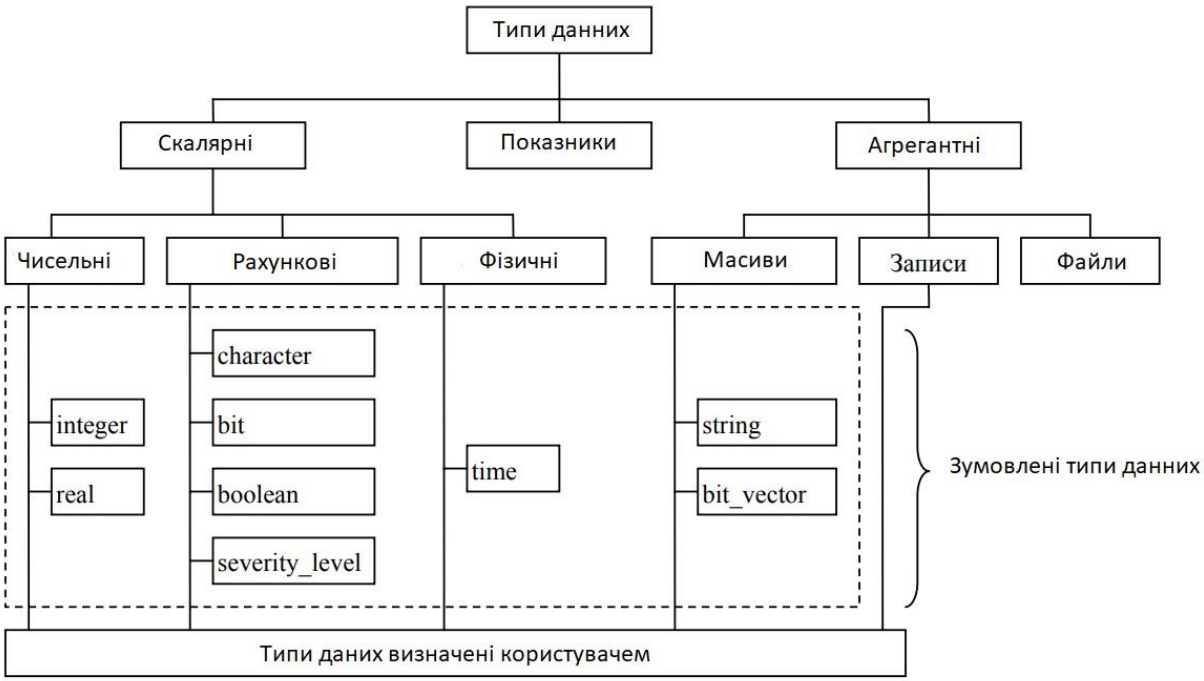


Рис. 2.1 Класифіккація типів данних в VHDL

Особливим відмінністю VHDL від конкуруючого Verilog є стандарт IEEE 1076 який містить в собі як загальні математичні пакети, так і спеціальні пакети для роботи з плаваючою комою.

Метою IEEE 1076 являється включення можливостей, які підвищують корисність мови за його прямим призначенням, а також розширюють його для обліку методологій перевірки дизайну, розроблених в промисловості. Ці нові можливості проектування і перевірки необхідні для того, щоб VHDL залишався актуальним і цінним для використання при розробці та верифікації електронних систем.

Включення раніше окремих, але пов'язаних стандартів спрощує ведення специфікацій. Цей стандарт переглядає і розширює довідкове керівництво по мові VHDL включаючи стандартну специфікацію інтерфейсу мови C; специфікації з раніше окремих, але пов'язаних стандартів IEEE Std тисячі сто шістьдесят чотири -1993,1 IEEE Std 1076.2 -1996 і IEEE Std 1076.3-1997; і загальні поліпшення мови в області проектування і перевірки електронних систем.

2.4. Огляд САПР Vivado

Vivado – це система автоматичного проектування від компанії Xilinx. В основі інтерфейсу лежить підхід, випробуваний в IDE PlanAhead і орієнтований в першу чергу на аналіз характеристик проекту і планування топології. Такий стиль проектування дозволяє розробникові зосередитися на вирішенні основних проблем, що виникають при роботі з ПЛІС великого логічного обсягу. Можна коротко перерахувати, що за проблеми маються на увазі.

У процесі отримання конфігурації ПЛІС програмне забезпечення формує список зв'язків між окремими компонентами, а потім намагається розмістити ці компоненти на кристалі і сформувати програмовані зв'язки між ними. Це типова комбінаторна задача виду «розмістити M елементів в N можливих позиціях», яка дуже важка для автоматичного рішення. Складність полягає в тому, що прямий перебір варіантів для досить великих M і N неможливий (а для ПЛІС серії Virtex-7 N обчислюється вже мільйонами), тому доводиться обмежуватися оптимізацією окремих випадків в поєднанні з евристичними алгоритмами. При оптимізації розміщення використовується алгоритм градієнтного спуску, відомим властивістю якого є висока чутливість до початкових умов. Для проектів на ПЛІС це означає, що ефективність розміщення істотно залежить від того, наскільки грамотно розробник задав проектні обмеження, що управляють топологією проекту «в цілому».

Іншою важливою властивістю Vivado є можливість управління всім циклом розробки за допомогою скриптового мови Tcl. Крім можливості копіювання будь-яких дій, включаючи додавання модулів і запуск основних процесів, Tcl лежить в основі нового формату опису проектних обмежень xdc (Xilinx Design Constraints). Цей формат замінив використовувався раніше формат ucf і володіє в порівнянні з ним більш гнучкими можливостями опису проектних обмежень, що полегшують побудову масштабованих проектів.

Вибір Vivado зумовлений не лише зручністю інтерфейсу для розробки, а й підтримкою усіх нині існуючих, та актуальних ПЛІС від компанії Xilinx.

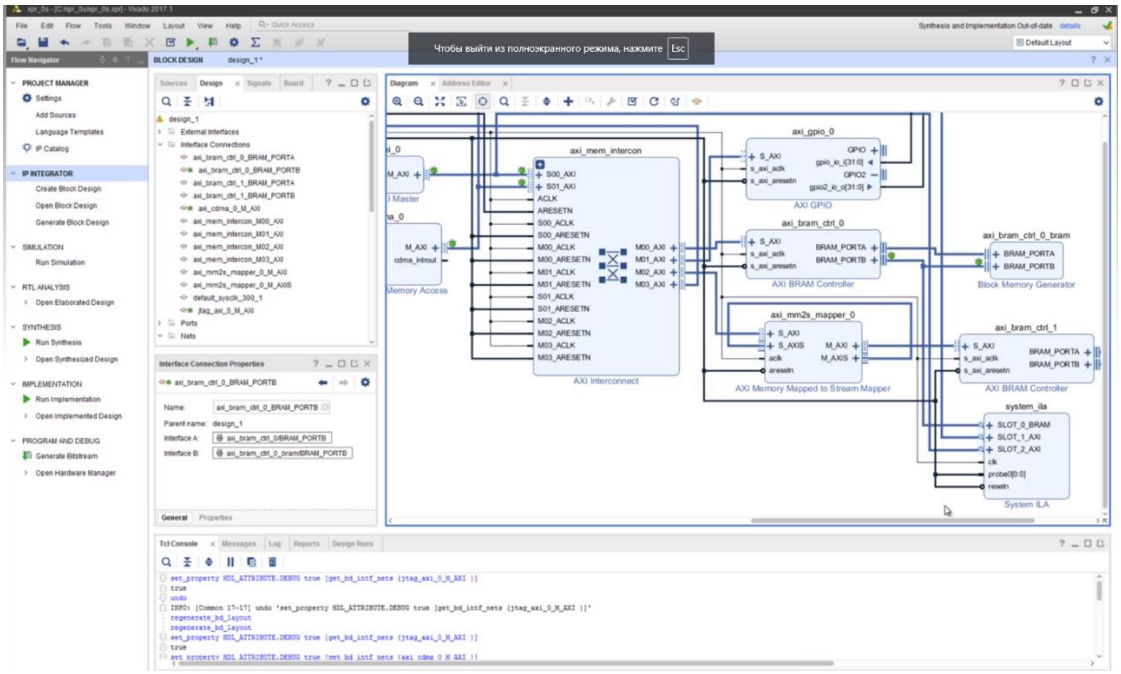


Рис.2.2 Інтерфейс САПР Vivado

Симулятор Vivado є компонентом дизайнерського набору Vivado. Це симулятор мови компіляції, який підтримує змішану мову, скрипти TCL, зашифрований IP та розширену перевірку.

Інтегратор Vivado IP дозволяє інженерам швидко інтегрувати та конфігурувати IP з великої бібліотеки IP Xilinx.

2.5. **Огляд сучасних сімейств ПЛІС від Xilinx**

ПЛІС серії Xilinx 7 включають в себе чотири сімейства FPGA, які задовольняють всьому спектру системних вимог: від низької вартості, малого форм-фактора, чутливих до вартості додатків з великими обсягами до ультрависокої пропускною здатності підключення, логічної ємності і можливостей обробки сигналів. для найвимогливіших високопродуктивних додатків. ПЛІС серії 7 включають в себе:

Сімейство Spartan-7: оптимізовано для низької вартості, мінімальної потужності і високої продуктивності вводу-виводу. Доступний в недорогий, дуже маленькою форм-факторі упаковці для найменшої площі друкованої плати.

Сімейство Artix-7: оптимізовано для додатків з низьким енергоспоживанням, що вимагають послідовних приймачів і високою DSP і логічної пропускної здатності. Забезпечує найнижчу загальну вартість матеріалів для високопродуктивних і чутливих до вартості додатків.

Сімейство Kintex-7: оптимізовано для кращої ціни і продуктивності з поліпшенням в 2 рази в порівнянні з попереднім поколінням, що забезпечує новий клас FPGA.

Сімейство Virtex-7: оптимізовано для забезпечення максимальної продуктивності і ємності системи зі збільшенням продуктивності системи в 2 рази. Пристрої з найвищими можливостями, реалізовані за допомогою технології стекового міжз'єднання.

Табл. 2.3 Порівняння сімейств 7 серії

	Spartan-7	Artix-7	Kintex-7	Virtex-7
Логічні елементи	102K	215K	478K	1,955K
Оперативна памьять	4.2 Mb	13 Mb	34 Mb	68 Mb
DSP-фрагменти	160	740	1,920	3,600
Продуктивність DSP	176 GMAC/s	929 GMAC/s	2,845 GMAC/s	5,335 GMAC/s
MicroBlaze CPU	260 DMIPs	303 DMIPs	438 DMIPs	441 DMIPs
Трансивери	–	16	32	96
Швидкість трансиверів	–	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s
Послідовна пропускна здатність	–	211 Gb/s	800 Gb/s	2,784 Gb/s
Інтерфейс PCIe	–	x4 Gen2	x8 Gen2	x8 Gen3
Інтерфейс пам'яті	800 Mb/s	1,066 Mb/s	1,866 Mb/s	1,866 Mb/s
Штифти вводу / виводу	400	500	500	1,200
Напруга вводу / виводу	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V

Зупинимося на сімействі Artix-7, вибір саме цього сімейства зумовлений наявністю інтерфейсу підтримки PCIe та трохи залишковою

потужністю.

Для даного проекту, розгляд сімейства Spartan-7 відкинуто через відсутність цього порту, який надасть необхідну швидкість передачі даних між системою та модулем, який в цю систему буде вживлений.

Табл. 2.4 Характеристики мікропроцессорів сімейства Artix-7

Плата	Логічні елементи	Логічні блоки для налаштування		Блоки RAM (Кб)			CIe	TPs	ADC блоки	Всього банків вводу-виводу	Максимальна кількість користувацьких штивтів вводу-виводу
		Фрагменти	Оперативна пам'ять (Кб)	8	6	MAX					
XC7A12T	12,800	2,000	171	0	0	720				3	150
XC7A15T	16,640	2,600	200	0	5	900				5	250
XC7A25T	23,360	3,650	313	0	5	1,620				3	150
XC7A35T	33,280	5,200	400	00	0	1,800				5	250
XC7A50T	52,160	8,150	600	50	5	2,700				5	250
XC7A75T	75,520	11,200	892	10	05	3,780				6	300
XC7A100T	101,440	15,850	1,188	70	35	4,860				6	300
XC7A200T	215,360	33,650	2,888	30	65	13,140		6		10	500

З характеристик плат бачимо що навіть наймолодший представник сімейства Artix-7 має 2000 налаштовуємих елементів, та оперативну пам'ять в 171 кілобайт, цього більш ніж вистачить для розрахунку квадратного кореня з чисел розміром до 32 біта.

2.6. **Огляд ПЛІС Artix-7 AC701**

Плата пропонує функції, загальні для багатьох вбудованих систем обробки, включаючи DDR3 пам'ять SODIMM, 4-полосний інтерфейс PCI Express, Ethernet PHY, штивти для вводу-виводу та інтерфейс UART. Сама плата має наступний вигляд:

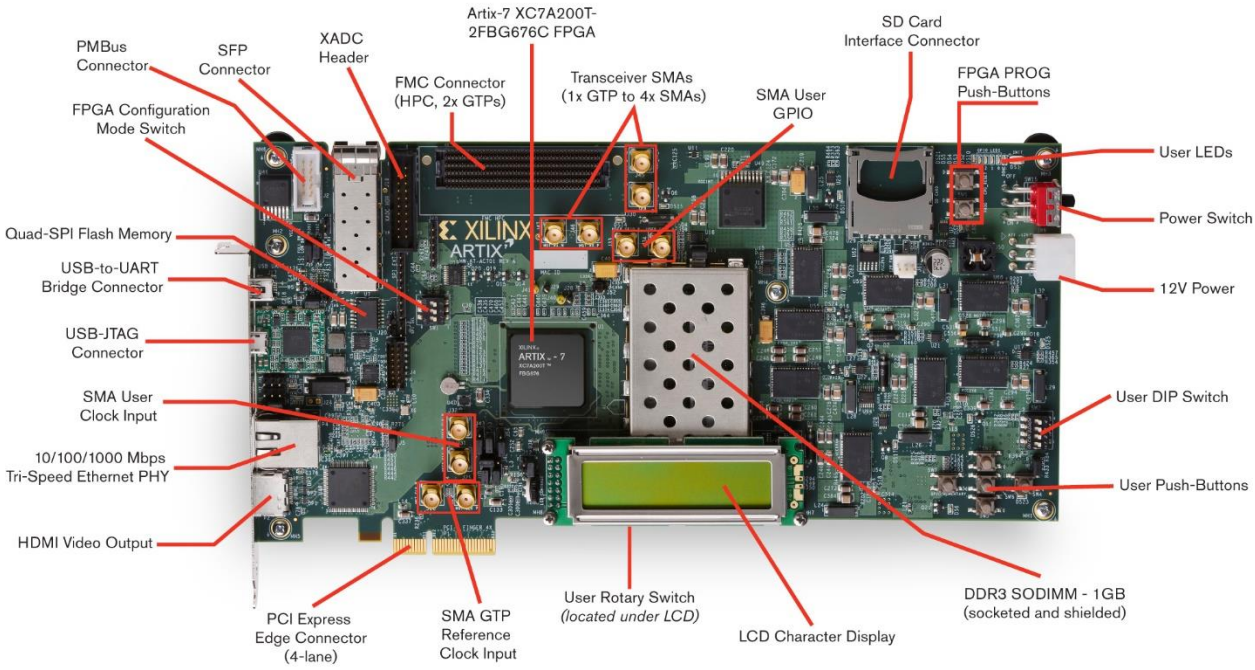


Рис. 2.3 ПЛІС Artix-7 AC701

На ній розміщенні всі необхідні інтерфейси вводу-виводу, які можуть знадобитися в ході розробки, а також ця плата підтримує підключення додаткових інтерфейсів при необхідності.

Структурно плата має наступний вигляд:

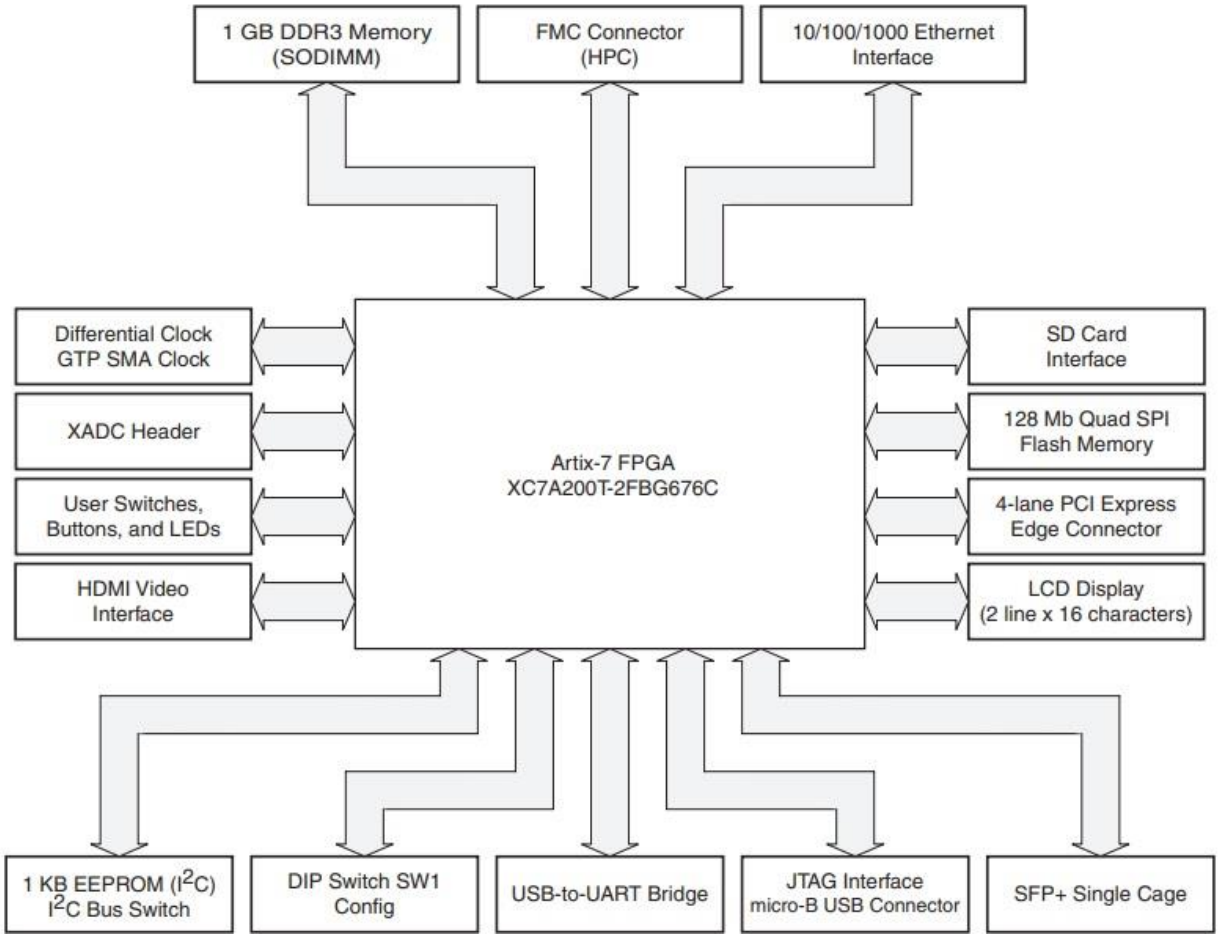


Рис.2.4 Структурна діаграма ПЛІС Artix-7 AC701

З основних переваг можна виділити такі як Фіксований генератор з диференціальним виходом 200 МГц, використовується як «системний» годинник для ПЛІС.

Програмований користувачем диференційний генератор (діапазон: 10-810 МГц, за замовчуванням 156,250 МГц).Диференціальний тактовий вхід SMA. Диференціальний вхід тактової частоти SMA GTP.

Джиттер для підтримки додатків CPRI / OBSAI, які виконують відновлення синхронізації з наданого користувачем модуля SFP.

Плата також розрахована на засоби проектування IP, а також включає в себе наступні типи пам’яті:

- DDR3 SODIMM 1 ГБ до 533 МГц / 1066 Мбит / с
- Quad SPI Flash: 32 МБ (256 МБ)
- IIC EEPROM: 8Kb
- Слот для SD-карти

Ця плата і буде являтися апаратною базою для розроблюваного проекту

ВИСНОВКИ ДО РОЗДІЛУ 2

Аналіз засобів для розробки обчислювальної системи на базі ПЛІС показав, що для побудови доцільно використовувати програмне середовище Vidvdo, та мову програмування VHDL. Це дозволить розробити програму досить швидко та якісно, завдяки промисловим світовим стандартам.

В якості апаратної бази було обрано ПЛІС Artix-7 AC701 , на базі мікропроцесорів XC7A12T.

РОЗДІЛ 3.

Представлення чисел з плаваючою комою та алгоритми у ПЛІС

3.1 Представлення чисел з плаваючою комою у ПЛІС

Сучасна елементна база на основі програмованих логічних інтегральних схем або Programmable Logic Devices типу FPGA (Field Programmable Gate Array), а також нові технології проектування з використанням відповідних інструментальних засобів дозволяють реалізувати в кристалі практично будь-який проект цифрового пристрою в необхідні терміни, маючи в наявності лише персональний комп'ютер і САПР ПЛІС. Істотним перевагою ПЛІС в порівнянні з замовними великими інтегральними схемами є їх універсальність і можливість швидкого програмування та перепрограмування під заданий проект.

Розробка цифрових пристроїв на ПЛІС істотно спрощується за рахунок використання готових технічних рішень, які називаються IP (Intellectual Property), - блоками, ядрами . IP-блоки є модулями, параметри яких настроюються користувачем під конкретні вимоги розроблюваного проекту. В даний час підготовка таких технічних рішень багаторазового використання сформувалася в окрему область діяльності, здійснювану рядом фірм для різних класів цифрових пристроїв. Цьому в чималому мірою сприяло впровадження в практику проектування мов високого рівня для опису апаратури, а також відповідних галузевих стандартів.

При вирішенні ряду задач доцільно використання формату представлення чисел з плаваючою точкою замість формату з фіксованою точкою, що володіє такими недоліками, як обмеження діапазону уявлення і втрата точності при розподілі двох великих чисел.

Функціональні IP-блоки з плаваючою точкою знаходять широке застосування при побудові математичних співпроцесорів, , вбудованих арифметичних співпроцесорів. У зв'язку з цим рядом фірм розробляються власні IP-блоки. Наприклад, фірма Nallatech розробила ядра для обробки 32-

розрядних операндів з плаваючою точкою згідно стандарту IEEE-754 під кристали серії Virtex. Однак такі ядра є комерційним продуктом і не підлягають тиражуванню.

Проблема проектування арифметичних пристроїв і алгоритмів для обробки операндів в форматі з плаваючою точкою при реалізації операцій додавання, множення і ділення актуальна і в даний час, що підтверджується доповідями по даній темі на таких представницьких симпозіумах, як Symposium on Field programmable gate arrays.

Стандарт IEEE-754 дає найбільш загальне уявлення для чисел з плаваючою точкою в сучасних комп'ютерах, включаючи Intel PC, Macintosh і більшість Unix платформ. Пропонується підхід до розробки пристрою на основі сучасних кристалів ПЛІС, що здійснює виконання операції ділення відповідно до стандартом IEEE-754, з використанням сучасних інструментальних засобів для проектування, моделювання і верифікації.

У стандарті IEEE-754 операнди в форматі одинарної точності представлені 32 бітами - 1 біт для знака, 8 біт для порядку і 23 біта для дробу мантиси. У зв'язку з тим, що даний формат передбачає наявність «прихованого» старшого біта, мантиса насправді містить 24 біта.

Число з плаваючою точкою представляється в такий спосіб:

$$X = (-1)^S * 2^{E-127} * 1, M$$

де S - знак числа; E - показник порядку (зміщений на b = +127);

M -нормалізована мантиса (M ∈) 2,1 [).

На рисунку наведено формат операнда і діапазон допустимих значень.

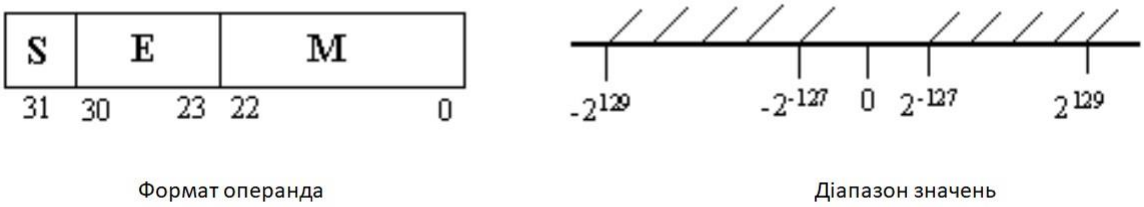


Рис.3.1 Формат операнда та діапазон значень

Стандарт IEEE-754 резервує ряд спеціальних значень:
машинний нуль "x_00000000_00000000000000000000000";
ненормалізоване число "x_00000000_xxxxxxxxxxxxxxxxxxxxxx1";
+ нескінченність "0_11111111_00000000000000000000000";
нескінченність "1_11111111_00000000000000000000000";
QNAN "x_11111111_1xxxxxxxxxxxxxxxxxxxxxx";
SNAN "x_11111111_0xxxxxxxxxxxxxxxxxxxxx1".

Тут x може приймати значення нуля або одиниці. NaN (Not a Number - не число) використовується для представлення значення, яке не представляється дійсним числом. NaN містить дві категорії: QNAN (Quiet NaN) і SNAN (Signaling NaN). QNAN – семантично позначає невизначені операції, SNAN - неприпустимі операції.

На мові VHDL використання стандарту IEEE-754 можна продемонструвати у написаному суматорі :

```
module FMULT_module(inA, inB, outY, clc);
input [31:0] inA;
input [31:0] inB;
output [31:0] outY;
input clc;
reg [31:0] outY;
reg [8:0] Exponenta;
reg [8:0] Exponenta1;
reg [7:0] ExpS1;
reg [7:0] ExpS2;
reg [47:0] Mantisa;
reg HbitA;
reg HbitB;
reg Hbit;
reg bitMaxA;
```

```
reg bitMinA;
reg bitMaxB;
reg bitMinB;
reg Underflow;
reg Overflow;
reg SigExp;
/*-----------------------------------

bit 31 - S

bits 30:23 - EXP
bits 22:0 - MANTISA
X=(-1)^S * 2^(EXP-127) * 1.MANTISSA

-----------------------------------*/

always @(posedge clc)
begin
outY[31] = inA[31] ^ inB[31]; //signed of member
ExpS1=inA[30:23]-8'd127;
ExpS2=inB[30:23]-8'd127;

Exponenta=$signed(ExpS1)+$signed(ExpS2);
Hbit = Overflow | Underflow;
Mantisa=$unsigned({1'b1,inA[22:0]}) * $unsigned({1'b1,inB[22:0]});
SigExp=Exponenta[8];
Exponenta1=Exponenta[7:0] ^ 8'hff;
Exponenta1=Exponenta1 + 1;
if(SigExp)
```

```
begin
if(Exponenta1[7:0]>8'd127)
begin
Exponenta=8'h81;
outY[22:0]=23'h0;
end
else
case(Mantisa[47:46])
2'b01 :
begin
outY[22:0]={Mantisa[45:23]};
end
2'b10 :
begin
outY[22:0]={Mantisa[45:23]};
outY[22:0]={1'b0,Mantisa[45:24]};
Exponenta=Exponenta + 8'h01;
end
2'b11 :
begin
outY[22:0]={Mantisa[46:24]};
Exponenta=Exponenta + 8'h01;
end
endcase
end
else
begin
if(Exponenta[7:0]>8'd127)
begin
```

```
        Exponenta=8'h80;
        outY[22:0]=23'h0;
        end
        else
        case(Mantisa[47:46])
        2'b01 :
        begin
        outY[22:0]={Mantisa[45:23]};
        end
        2'b10 :
        begin
        outY[22:0]={1'b0,Mantisa[45:24]};
        Exponenta=Exponenta + 8'h01;
        end
        2'b11 :
        begin
        outY[22:0]={Mantisa[46:24]};
        Exponenta=Exponenta + 8'h01;
        end
        endcase
        end
        Exponenta=$signed(Exponenta)+$signed(8'd127);
        outY[30:23]=Exponenta[7:0];
        end
    endmodule
```

3.2. Алгоритми для обчислення квадратного кореня на ПЛІС

Функція квадратного кореня широко використовується в комп'ютерній графіці, зображення та тон обробки, статистика, додатки зв'язку та наукових розрахунків.

Через складності, пов'язаних з реалізацією алгоритмів квадратного кореня, його проектування в цифровій системі завжди було вузьким місцем. Базові операції, такі як додавання і віднімання, легко реалізувати в ПЛІС, тому що інструменти синтезу оптимізували одиниці складання та віднімання на основі архітектури ПЛІС.

Множення, ділення і квадратний корінь є складними операціями. Квадратний корінь обчислювально інтенсивний, оскільки він включає в себе методи збіжності і наближення. Багато алгоритмів та методів були розроблені для реалізації цього на ПЛІС. Але не всі вони досить ефективні з точки зору часу, швидкості і площі всередині кристалу.

CORDIC є акронімом для Координатного Комп'ютера Обертання. Основний механізм алгоритму CORDIC відрізняється від більшої частини обладнання, ефективного алгоритму, який потребує ітеративних операцій shift-add. Алгоритм CORDIC виділяє від необхідних явних множинників і підходить для виведення множинних функцій, таких як синус, косинус, тангенс, арктангенс, векторну величину, квадратний корень, гіперболічні та логарифмічні функції.

Алгоритм CORDIC вимагає наступних операцій:

1 пошук по таблиці на ітерацію

2 зсув на ітерацію

3 складання на ітерацію

Для гіперболічних основних алгоритмів CORDIC, таких як квадратний корінь, конкретні ітерації (i = 4, 13, 40, 121 ..., k, 3k + 1 ...) повторюються, щоб досягти результатів результатів.

Розрахунок квадратного корня Використовує гіперболічне ядро

векторизації CORDIC розумний вибір початкових значень можливостей ядру CORDIC дозволяє вивести квадратний корінь.

По перше, виконуючи шагі ініціалізації виконуються:

1)x_0 виставлен у $v + 0.25$

2)y_0 виставлен у $v - 0.25$

Після N ітерацій ці початкові значення приводять до наступного виходу як:

$$x_N \approx A_N\sqrt{(v + 0.25)^2 + (v - 0.25)^2}$$

Це можна спростити до вигляду:

$$x_N \approx A_N\sqrt{v}$$

де A_N посилення CORDIC.

Альтернативний опис алгоритму CORDIC може виглядати наступним чином. Алгоритм зводиться до повторення одноманітних ітерацій, результатами яких є чергові точні цифри результату. Для кожного числа x ∈[0.25; 1.0] знаходять коефіцієнти a_i∈ [0; 1], такі що

$$x\prod_{i=1}^{n}(1 + a_i 2^{-i})^2 \approx \sqrt{x}$$

Отримаємо наступний алгоритм:

```
x[0] = x; y[0] = x;
for(i = 0, i < n, i++) {t = x[i] + 2^(-i)*x[i];
        q = t + 2^(-i)*t;
        if (q < 1) {x[i+1] = q; y[i+1] = y[i] + 2^(-i)*y[i];}// a[i]=1
        else {x[i+1] = x[i]; y[i+1] = y[i];}// a[i]=0
 }
```

Результат : $\sqrt{x} \approx$ y[n]

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі було розглянуто представлення чисел з плаваючою комою у ПЛІС.

Було розглянуто стандарт IEEE-754 для чисел з плаваючою комою, а також представлення використання цього стандарту на прикладі суматору написаного мовою VHDL

Було обрано алгоритм на основі CORDIC для розрахунку квадратного кореня на ПЛІС.

РОЗДІЛ 4.

Моделювання пристрою

Одна з причин вибору алгоритму та плати це програмне ядро Xilinx CORDIC.

Xilinx CORDIC це універсальне програмне ядро для розрахунків на базі алгоритму Xilinx CORDIC.У САПР Vivado доступний інструмент Xilinx Core Generator, завдяки якому можна додати в наш проект потрібне нам ядро.

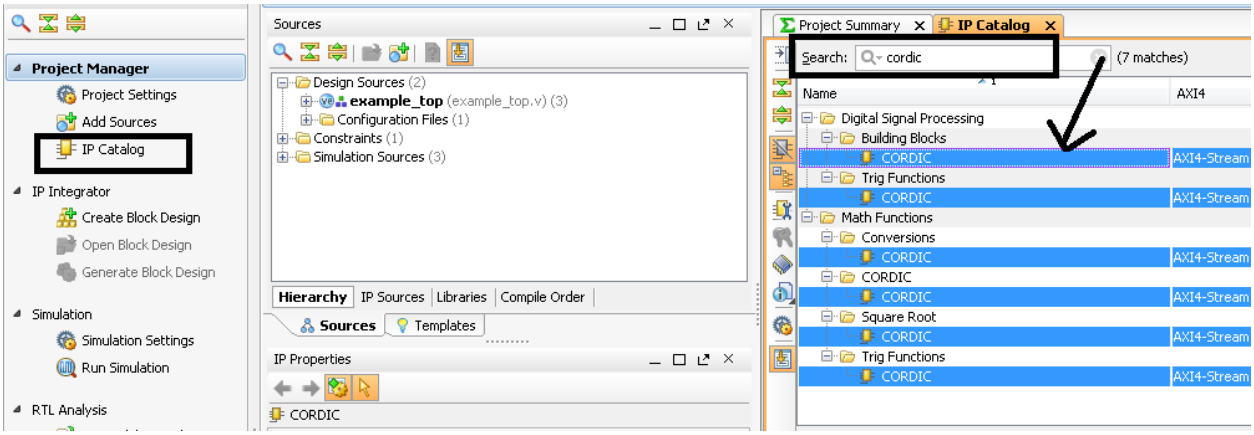


Рис.4.1 Вибір ядра CORDIC

У цьому ядрі для підготовки описів пристроїв, що виконують операції вилучення квадратного кореня з значень, представлених на вхідний шині даних, є VHDL-опис елемента square_root_cordic, модифікований текст якого має наступний вигляд:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- synthesis translate_off
Library XilinxCoreLib;
-- synthesis translate_on
ENTITY square_root_cordic IS
    port (
        x_in: IN std_logic_VECTOR(47 downto 0);
        nd: IN std_logic;
        x_out: OUT std_logic_VECTOR(47 downto 0);
        rdy: OUT std_logic;
        clk: IN std_logic;
        ce: IN std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic
        );
END square_root_cordic;
--
ARCHITECTURE square_root_cordic_a OF square_root_cordic IS
-- synthesis translate_off
component wrapped_square_root_cordic
    port (
        x_in: IN std_logic_VECTOR(47 downto 0);
        nd: IN std_logic;
        x_out: OUT std_logic_VECTOR(47 downto 0);
        rdy: OUT std_logic;
        clk: IN std_logic;
        ce: IN std_logic;
        aclr: IN std_logic;
        sclr: IN std_logic
        );
end component;
--
```

Рис. 4.2 Виклик ядра CORDIC

```
        generic map(
                        c_has_clk => 1,
                        c_has_x_out => 1,
                        c_has_y_in => 0,
                        c_reg_inputs => 1,
                        c_architecture => 2,
                        c_input_width => 48,
                        c_iterations => 0,
                        c_precision => 0,
                        c_has_rdy => 1,
                        c_has_sclr => 1,
                        c_has_nd => 1,
                        c_scale_comp => 0,
                        c_enable_rlocs => 1,
                        c_has_phase_in => 0,
                        c_has_rfd => 0,
                        c_cordic_function => 6,
                        c_has_ce => 1,
                        c_mif_file_prefix => «cor1»,
                        c_round_mode => 1,
                        c_has_aclr => 1,
                        c_sync_enable => 1,
                        c_has_y_out => 0,
                        c_data_format => 1,
                        c_reg_outputs => 1,
                        c_coarse_rotate => 0,
                        c_phase_format => 0,
                        c_has_phase_out => 0,
                        c_has_x_in => 1,
                        c_pipeline_mode => -1,
                        c_output_width => 48
                        );
-- synthesis translate_on
BEGIN
-- synthesis translate_off
U0 : wrapped_square_root_cordic
      port map (
                        x_in => x_in,
                        nd => nd,
                        x_out => x_out,
                        rdy => rdy,
                        clk => clk,
                        ce => ce,
                        aclr => aclr,
                        sclr => sclr
                        );
-- synthesis translate_on
--
END square_root_cordic_a;
```

Рис. 4.3 Опис елемента square_root_cordic

У сформованому елементі square_root_cordic використовуються 48-розрядні вхідна і вихідна шини. До складу архітектури елемента square_root_cordic включені вхідні і вихідні регістри.

Вибір оптимального варіанту конвеєрної організації виконання операцій в цьому елементі забезпечує досить високу продуктивність обчислень квадратного кореня без залучення додаткових таблиць перетворення ПЛІС. В елементі square_root_cordic задіяні входи сигналів синхронного і асинхронного сбросу, а також вхід сигналу дозволу синхронізації. Сигнал на вході синхронного сбросу має більш високий пріоритет, ніж сигнал на вході дозволу синхронізації.

До складу інтерфейсу елемента, що розглядається також включений вхід сигналу New Data, що підтверджує достовірність вхідних даних, і вихід сигналу Ready, який повідомляє про готовність достовірних результатів обчислень квадратного кореня.

Опис кожного примірника компонента square_root_cordic в складі блоку визначення архітектури розроблювального пристрою виконується за допомогою оператора, шаблон якого має наступний вигляд:

```
square_root_cordic
    port map (
                x_in => x_in,
                nd => nd,
                x_out => x_out,
                rdy => rdy,
                clk => clk,
                ce => ce,
                aclr => aclr,
                sclr => sclr
                );
```

Рис. 4.4 Опис примірника

Оцінка обсягу різних ресурсів кристала, необхідних для автономної реалізації елемента square_root_cordic, наведена в інформаційній панелі, вигляд якої представлений на рисунку:

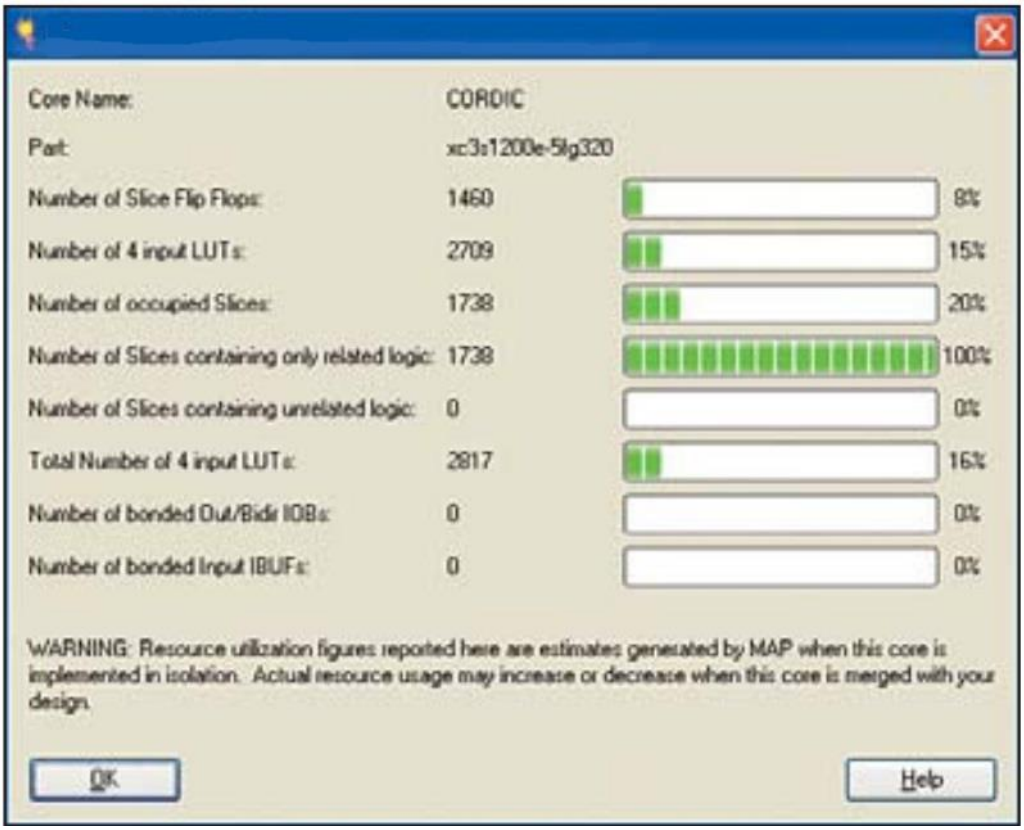


Рис.4.5 відомості про ресурси ПЛІС, використовуваних для реалізації елемента square_root_cordic.

Приклад роботи модулю можна спостерігати на рисунку:

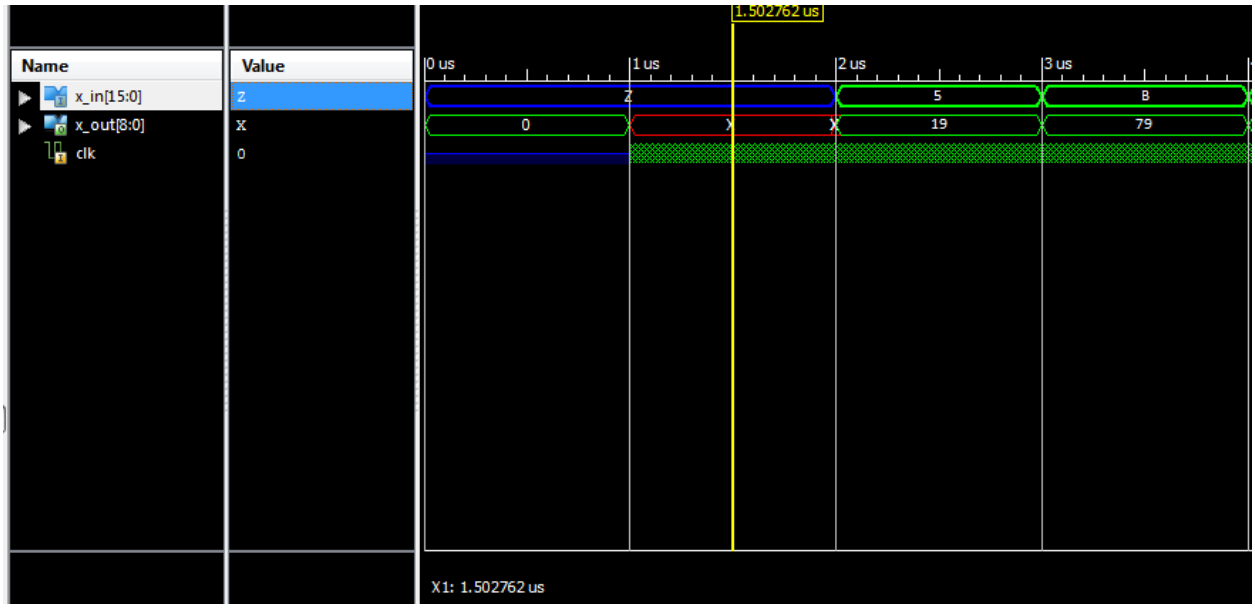


Рисунок4.6 Робота модулю

Де x дорівнює вхідному значенню, а z кінцевому.

ВИСНОВКИ ДО РОЗДІЛУ 4

В даному розділі було проведено моделювання пристрою для обчислення квадратного кореня з плаваючою комою.

Було модифіковано програмне ядро Xilinx CORDIC.

Було показана робота модулю з вирахування квадратного кореня.

ВИСНОВКИ

У представленій роботі було розглянуті сімейства ПЛІС, мови проектування на ПЛІС.

Були розглянуті представлення чисел з плаваючою комою, а також алгоритми для знаходження квадратного кореня, які часто використовуються на ПЛІС.

Було проведено моделювання модулю на ПЛІС для знаходження квадратного кореня чисел з плаваючою комою. Такі модулі суттєво скорочують час на розрахунки, та прискорюють роботу системи, у яку вони вживлені.

СПИСОК ВИКОРИСТАННОЇ ЛІТЕРАТУРИ

1. Огляд архітектур FPGA.Статья [Режим доступу – Електронне джерело: https://www.terraelectronica.ru/news/5720]

2. Програмовані логічні інтегральні схеми.Сатья.[Режим доступу – Електронне джерело: https://portal.tpu.ru/SHARED/g/]

3. Класифікація ПЛИС. Статья. .[Режим доступу – Електронне джерело: http://digteh.ru/digital/PLD/]

4. ПЛМ ПЛІС. Курс лекцій. .[Режим доступу – Електронне джерело: http://www.studfiles.ru/preview/5863387/]

5. ПЛІС для початківців. Статья.[Режим доступу – Електронне джерело: http://we.easyelectronics.ru/plis/plis-zametki-nachinayuschego.html]

6. Архітектура FPGA. Статья. .[Режим доступу – Електронне джерело: https://marsohod.org/11-blog/265-fpga/]

7. Square root calculations in FPGA. Доповідь.Sergiyenko A. M., Hasan M. J., Sergiyenko P. A. «Computer Engineering Department of NTUU “Igor Sikorsky Kyiv Polytechnic Institute”, Kyiv, Ukraine» .[Режим доступу – Електронне джерело: https://kanyevsky.kpi.ua/wp-content/uploads/2018/10/square-root-calculations.pdf]

8. Розробка базових компонентів цифрових пристроїв.Доповідь. .[Режим доступу – Електронне джерело: https://www.kite.ru/assets/files/pdf/2008_3_80.pdf]

9.VHDL. wiki/ [Режим доступу – Електронне джерело: https://ru.wikipedia.org/wiki/VHDL]

10.Verilog. wiki. / [Режим доступу – Електронне джерело: https://ru.wikipedia.org/wiki/Verilog]

11. Методичні вказівки по курсу ЕОМ. Берчун Ю.В. [Режим доступу – Електронне джерело: http://window.edu.ru/resource/770/72770/files/VHDL.pdf]

12. 7 Series FPGAs Packaging and Pinout. Докуменатція. [Режим доступу – Електронне джерело:

https://sigma.octopart.com/138898721/technical_drawing/Xilinx-XC7A12T-1CSG325C.pdf]

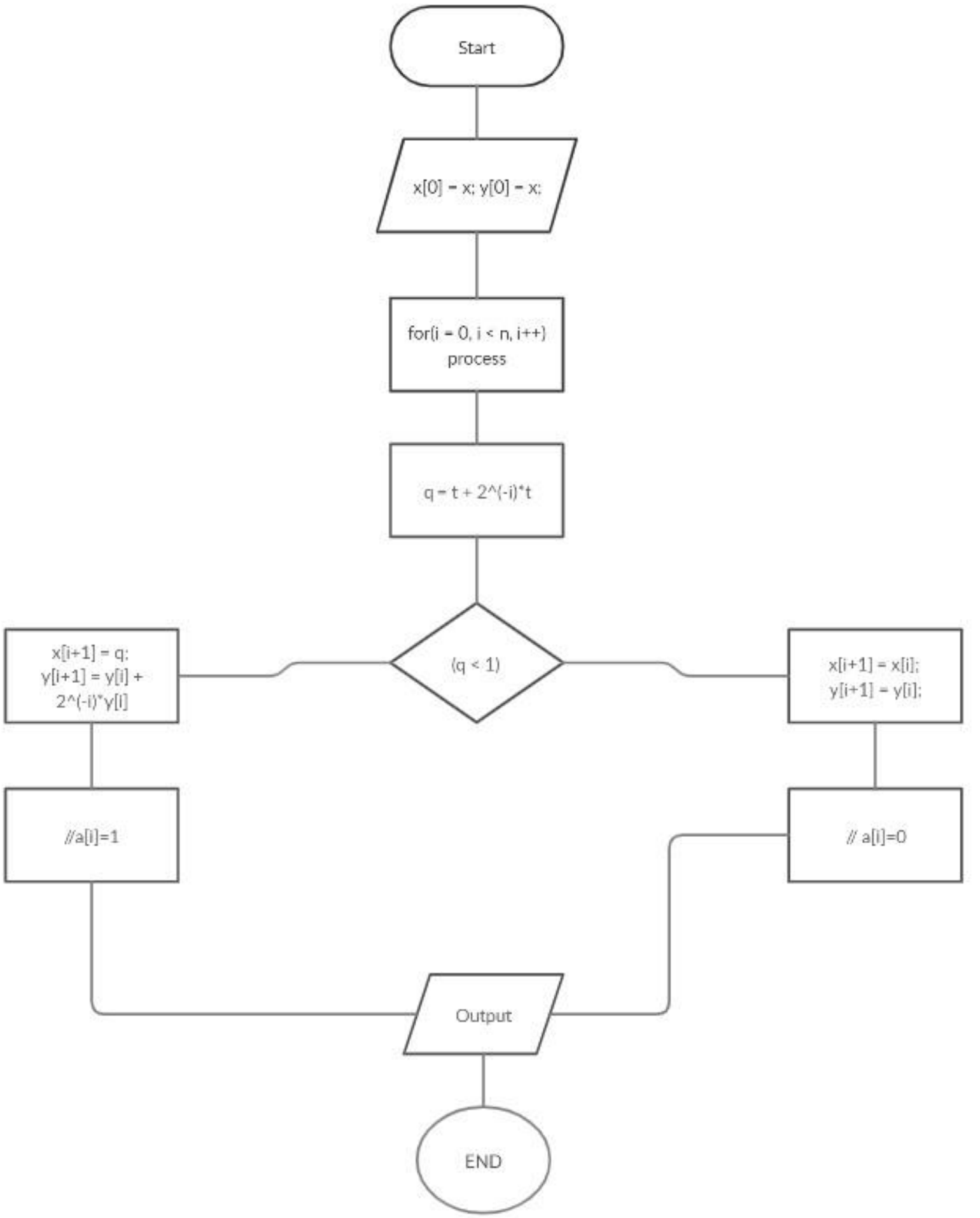
13. FPGA Implementation of Cordic Processor for Square Root Function.Наукова статья. [Режим доступу – Електронне джерело: https://www.ijarcce.com/upload/2016/november-16/IJARCCE%2015.pdf]

ДОДАТОК А

МОДУЛЬ ОБЧИСЛЕННЯ КВАДРАТНОГО КОРЕНЯ З ПЛАВАЮЧОЮ КОМОЮ

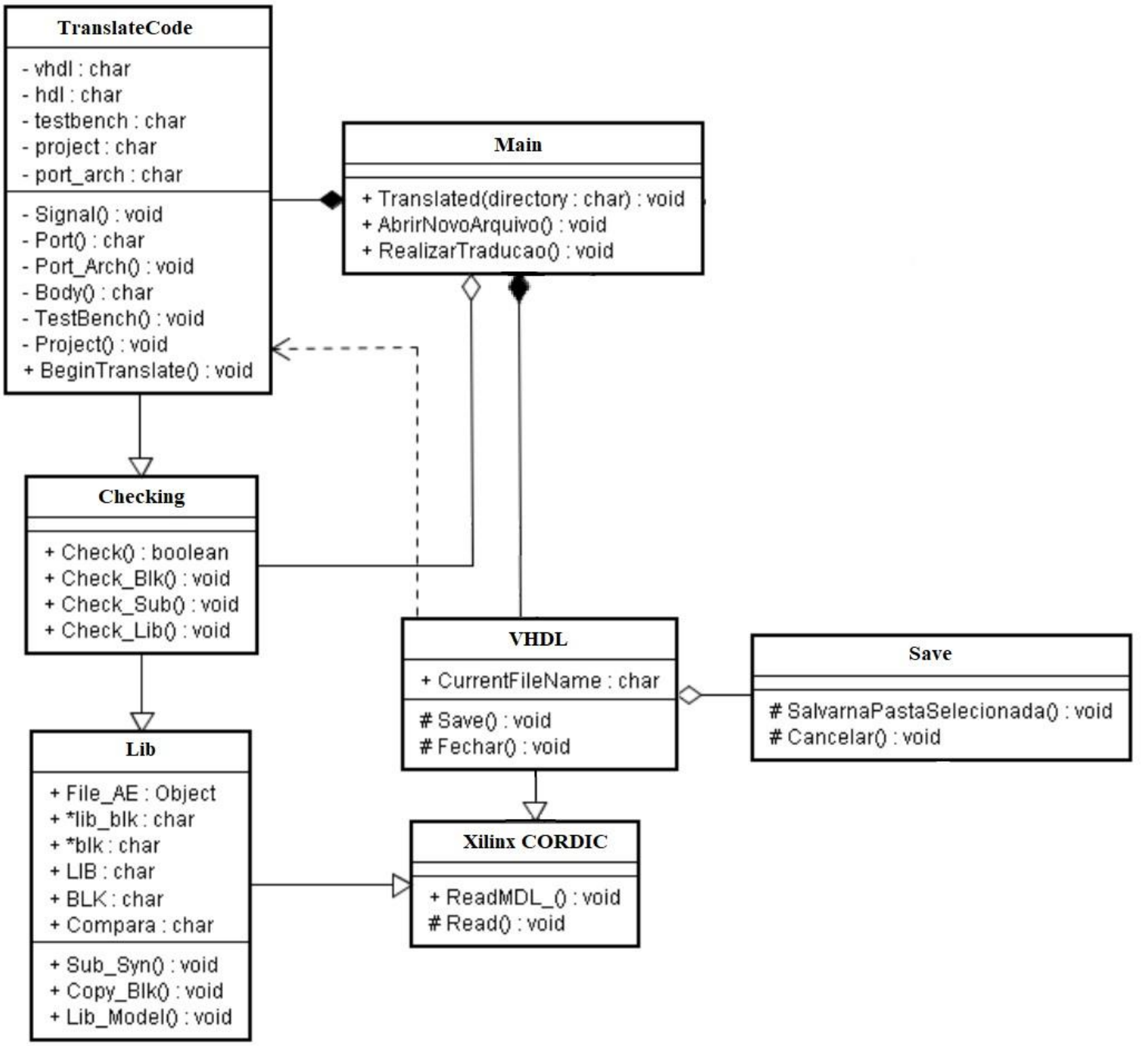
КОПІЇ ГРАФІЧНИХ МАТЕРІАЛІВ

Київ – 2020

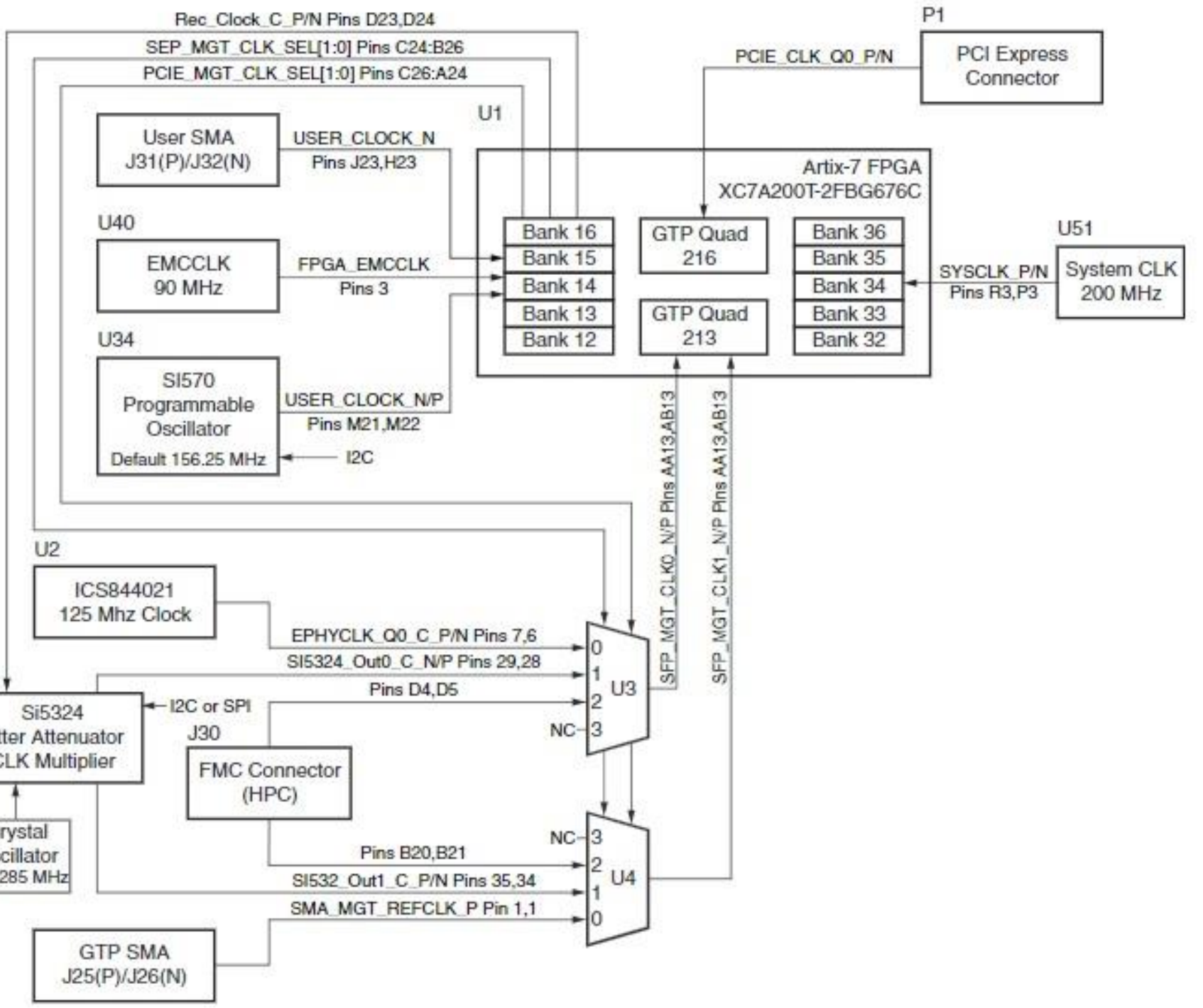


					ІАЛЦ. 467450.005 Д1
Зм.	Арк.	№ докум.	Підпис	Дата	
Розроб.		Трофимчук Д.А.			Модуль обчислення квадратного кореня з плаваючою комою ПРИНЦИПОВА СХЕМА
Перевір.		Сергієкно А.М..			
Н. контр.		Сімоненко В. П.			
Зав. каф.		Стіренко С. Г.			

Літ.	Аркуш	Аркушів
	1	1



					ІАЛЦ. 467450.005 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Трофимчук Д.А.			Модуль обчислення квадратного кореня з плаваючою комою ФУНКЦІОНАЛЬНА СХЕМА	Літ. / Аркуш 1 / Аркушів 1
Перевір.		Сергієкно А.М..				
Н. контр.		Сімоненко В. П.				
Зав. каф.		Стіренко С. Г.				



					ІАЛЦ. 467450.005 Д1
Зм.	Арк.	№ докум.	Підпис	Дата	
Розроб.		Трофимчук Д.А.			Модуль обчислення квадратного кореня з плаваючою комою СТРУКТУРНА СХЕМА
Перевір.		Сергієкно А.М..			
Н. контр.		Сімоненко В. П.			
Зав. каф.		Стіренко С. Г.			

Літ.	Аркуш	Аркушів
	1	1

ДОДАТОК Б

МОДУЛЬ ОБЧИСЛЕННЯ КВАДРАТНОГО КОРЕНЯ З ПЛАВАЮЧОЮ КОМОЮ

ЛІСТИНГ ПРОГРАМИ

Київ – 2020

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- use STD.textio.all;                      -- basic I/O
-- use IEEE.std_logic_textio.all;           -- I/O for logic types

entity sqrt is
    Port ( x : in  STD_LOGIC_VECTOR (31 downto 0);
           y : out  STD_LOGIC_VECTOR (31 downto 0));
end sqrt;

architecture Behavioral of sqrt is

    function bit2bit_sq(x: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is
        variable y : STD_LOGIC_VECTOR(2*x'left+1 downto 0);
    -- Returns x^2 by intercalating zeros in the argument,
    -- were x has only one bit different from zero.
    begin
        for i in x'left downto 0 loop
            -- x'right must be zero
            y(2*i):=x(i);
            y(2*i+1):='0';
        end loop;
        return y;
    end;

begin
    process(x)
        variable x_mantissa : STD_LOGIC_VECTOR (22 downto 0);
        variable x_exponent : STD_LOGIC_VECTOR (7 downto 0);
        variable x_sign : STD_LOGIC;
        variable y_mantissa : STD_LOGIC_VECTOR (22 downto 0);
        variable y_exponent : STD_LOGIC_VECTOR (7 downto 0);
        variable y_sign : STD_LOGIC;

        variable ix: STD_LOGIC_VECTOR (25 downto 0);
        variable a : STD_LOGIC_VECTOR (51 downto 0);
        variable biti : STD_LOGIC_VECTOR (25 downto 0);
        variable r : STD_LOGIC_VECTOR (51 downto 0);
        variable rt : STD_LOGIC_VECTOR (52 downto 0);

        -- variable my_line : line;  -- type 'line' comes from textio
   begin
        x_mantissa := x(22 downto 0);
        x_exponent := x(30 downto 23);
        x_sign := x(31);

        y_sign := '0';

        if (x_exponent="00000000") then -- zero
            y_exponent := (others=>'0');
```

```
            y_mantissa := (others=>'0');
        elsif (x_exponent="11111111") then -- infinity
            y_exponent := (others=>'1');
            y_mantissa := (others=>'0');
        else

            if (x_exponent(0)='1') then -- exponent-127 is even
                y_exponent := '0' & x_exponent(7 downto 1) + 64;
                ix := "01" & x_mantissa & '0';
            else -- exponent-127 is odd
                -- shift mantissa one to the left and subtract one from x_exponent
                y_exponent := '0' & x_exponent(7 downto 1) + 63;
                ix := '1' & x_mantissa & "00";
            end if;
            -- mantissa is m=ix/2^24
            -- (one zero was added to the right to make the power even)
            -- let the result of the integer square root algorithm be iy (26 bits)
            -- iy = sqtr(ix)*2^13
            -- resulting that sqrt(m)=iy/2^25

            -- Integer input N bits square root algorithm:
            -- r is be the reminder, r=ix-z^2, and z(N+1) the result,
            --  with bit(N)=1/2^(N/2), and bit(n)=2^(N/2-n)
            -- Test each bit in the result, from the most significative to the least
            -- significative: n goes from zero no N.
            -- if bit is one: r(n+1) = ix - (z(n)+bit(n))^2 =
            --                                 r(n) - 2z(n)bit(n) - bit(n)^2
            -- else           r(n+1) = r(n)
            -- bit will be one if the resulting remainder is positive.
            -- making a(n) = 2z(n)bit(n), one has,
            -- if bit is one: a(n+1) = 2(z(n)+bit(n))bit(n)/2 =
            --                                     a(n)/2+bit(n)^2
            -- else           a(n+1) = a(n)/2
            -- and a(N+1) = 2z(N+1)/2^(N/2+1) = z(N+1)/2^(N/2)

            -- VHDL Implementation

            a := (others=>'0');

            biti := "10" & x"000000"; -- 2^(25)
            -- biti has the bit being evaluated equal to one
            r(51 downto 26):= ix; -- r is in Q26
            r(25 downto 0):=(others=>'0');

            for i in 25 downto 0 loop
                rt := ('0' & r) - ('0' & (a or bit2bit_sq(biti)));
                -- trial for the new value for the reminder
                a := '0' & a(51 downto 1); -- srl
                if (rt(52)='0') then -- rt>=0
                    r := rt(51 downto 0);
                    a := a or bit2bit_sq(biti); -- the adder is safelly replaced by
an or
                end if;
```

```
                biti := '0' & biti(25 downto 1); -- srl 1
            end loop;

            a(24 downto 2) := a(24 downto 2)+a(1); -- round
            -- even for ix = all '1' a will not oveflow

            -- a is the result
            y_mantissa := a(24 downto 2);

        end if;

        y(22 downto 0) <= y_mantissa;
        y(30 downto 23) <= y_exponent


/

*
 * f15_logpwr.v
 *
 * Log Power computation
 * Take a complex 16 bits input and outputs a 16 bits estimate
 * of 2048 * log2(i^2+q^2).
 *
 * Fully-pipelined, 12 levels
 *
 * Copyright (C) 2014  Ettus Corporation LLC
 * Copyright 2018 Ettus Research, a National Instruments Company
 *
 * SPDX-License-Identifier: LGPL-3.0-or-later
 *
 * vim: ts=4 sw=4
 */

`ifdef SIM
`default_nettype none
`endif

module f15_logpwr(
        input  wire [15:0] in_real_0,
        input  wire [15:0] in_imag_0,
        output wire [15:0] out_12,
        input  wire [31:0] rng,
        input  wire [ 1:0] random_mode, /* [0] = lsb random ena, [1] = random add */
        input  wire clk,
        input  wire rst
);

        // Signals
                // Randomness control
        reg   [7:0] rng_lsb;
        wire  [6:0] opmode;
```

```
                // Power squared
        wire [47:0] dsp_pchain_3;
        wire [47:0] dsp_pout_4;

        wire  [4:0] msb_check;
        wire [31:0] pwr_4;
        reg  [31:0] pwr_5,  pwr_6,  pwr_7,  pwr_8,  pwr_9;
        reg   [4:0] log2_5, log2_6, log2_7, log2_8, log2_9;

                // LUT
        wire [15:0] lut_addr_9;
        wire [31:0] lut_do_11;

        wire        msb_9, msb_11;
        wire  [4:0] lsbs_9, lsbs_11;
        wire  [4:0] log2_11;

                // Final value
        reg  [20:0] final_12;


        // -------------
        // Power squared
        // -------------
                // Output is (in_real * in_real) + (in_imag * in_imag)
                // with possibly some random lsb filled in for in_{real,imag} and
some
                // noise added to the result.

        // Randomness control
        always @(posedge clk)
                if (random_mode[0])
                        rng_lsb <= rng[31:24];
                else
                        rng_lsb <= 8'h00;

        assign opmode = random_mode[1] ? 7'b0110101 : 7'b0000101;

        // Square of in_real + noise
        DSP48E1 #(
                .A_INPUT("DIRECT"),
                .B_INPUT("DIRECT"),
                .USE_DPORT("FALSE"),
                .USE_MULT("MULTIPLY"),
                .AUTORESET_PATDET("NO_RESET"),
                .MASK(48'h3fffffffffff),
                .PATTERN(48'h000000000000),
                .SEL_MASK("MASK"),
                .SEL_PATTERN("PATTERN"),
                .USE_PATTERN_DETECT("NO_PATDET"),
                .ACASCREG(1),
                .ADREG(0),
                .ALUMODEREG(1),
```

```
                .AREG(1),
                .BCASCREG(1),
                .BREG(1),
                .CARRYINREG(1),
                .CARRYINSELREG(1),
                .CREG(1),
                .DREG(0),
                .INMODEREG(1),
                .MREG(1),
                .OPMODEREG(1),
                .PREG(1),
                .USE_SIMD("ONE48")
        )
        dsp_real_sq_I (
                .PCOUT(dsp_pchain_3),
                .ACIN(30'h0000),
                .BCIN(18'h000),
                .CARRYCASCIN(1'h0),
                .MULTSIGNIN(1'h0),
                .PCIN(48'h000000000000),
                .ALUMODE(4'b0000),              // Z + X + Y + CIN
                .CARRYINSEL(3'h0),
                .CEINMODE(1'b1),
                .CLK(clk),
                .INMODE(5'b00000),              // B=B2, A=A2
                .OPMODE(opmode),         // X=M1, Y=M2, Z=(random_mode[1] ? C : 0)
                .RSTINMODE(rst),
                .A({{12{in_real_0[15]}}, in_real_0, rng_lsb[7:6]}),
                .B({                     in_real_0, rng_lsb[5:4]}),
                .C({{41{1'b0}},rng[6:0]}),
                .CARRYIN(1'b0),
                .D(25'h0000),
                .CEA1(1'b0),
                .CEA2(1'b1),
                .CEAD(1'b0),
                .CEALUMODE(1'b1),
                .CEB1(1'b0),
                .CEB2(1'b1),
                .CEC(1'b1),
                .CECARRYIN(1'b1),
                .CECTRL(1'b1),
                .CED(1'b0),
                .CEM(1'b1),
                .CEP(1'b1),
                .RSTA(rst),
                .RSTALLCARRYIN(rst),
                .RSTALUMODE(rst),
                .RSTB(rst),
                .RSTC(rst),
                .RSTCTRL(rst),
                .RSTD(rst),
                .RSTM(rst),
                .RSTP(rst)
```

```
);

// Square of in_imag and final sum
DSP48E1 #(
        .A_INPUT("DIRECT"),
        .B_INPUT("DIRECT"),
        .USE_DPORT("FALSE"),
        .USE_MULT("MULTIPLY"),
        .AUTORESET_PATDET("NO_RESET"),
        .MASK(48'h3fffffffffff),
        .PATTERN(48'h000000000000),
        .SEL_MASK("MASK"),
        .SEL_PATTERN("PATTERN"),
        .USE_PATTERN_DETECT("NO_PATDET"),
        .ACASCREG(1),
        .ADREG(0),
        .ALUMODEREG(1),
        .AREG(2),
        .BCASCREG(1),
        .BREG(2),
        .CARRYINREG(1),
        .CARRYINSELREG(1),
        .CREG(1),
        .DREG(0),
        .INMODEREG(1),
        .MREG(1),
        .OPMODEREG(1),
        .PREG(1),
        .USE_SIMD("ONE48")
)
dsp_imag_sq_I (
        .P(dsp_pout_4),
        .ACIN(30'h0000),
        .BCIN(18'h000),
        .CARRYCASCIN(1'h0),
        .MULTSIGNIN(1'h0),
        .PCIN(dsp_pchain_3),
        .ALUMODE(4'b0000),              // Z + X + Y + CIN
        .CARRYINSEL(3'h0),
        .CEINMODE(1'b1),
        .CLK(clk),
        .INMODE(5'b00000),              // B=B2, A=A2
        .OPMODE(7'b0010101),     // X=M1, Y=M2, Z=PCIN
        .RSTINMODE(rst),
        .A({{12{in_imag_0[15]}}, in_imag_0, rng_lsb[3:2]}),
        .B({                     in_imag_0, rng_lsb[1:0]}),
        .C(48'h0000),
        .CARRYIN(1'b0),
        .D(25'h0000),
        .CEA1(1'b1),
        .CEA2(1'b1),
        .CEAD(1'b0),
        .CEALUMODE(1'b1),
```

```
        .CEB1(1'b1),
        .CEB2(1'b1),
        .CEC(1'b1),
        .CECARRYIN(1'b1),
        .CECTRL(1'b1),
        .CED(1'b0),
        .CEM(1'b1),
        .CEP(1'b1),
        .RSTA(rst),
        .RSTALLCARRYIN(rst),
        .RSTALUMODE(rst),
        .RSTB(rst),
        .RSTC(rst),
        .RSTCTRL(rst),
        .RSTD(rst),
        .RSTM(rst),
        .RSTP(rst)
);

assign pwr_4 = dsp_pout_4[35:4];


// ---------------------------------
// Log2 computation and normalization
// ---------------------------------
        // When shifting, instead of zero filling, we fill with RNG data
        // Again, this helps reduce the visible quantization effects
        // for very low power values.

// First stage
assign msb_check[4] = |(pwr_4[31:16]);

always @(posedge clk)
begin
        if (msb_check[4])
                pwr_5 <= pwr_4;
        else
                pwr_5 <= { pwr_4[15:0], rng[31:16] };

        log2_5 <= { msb_check[4], 4'b0000 };
end

// Second stage
assign msb_check[3] = |(pwr_5[31:24]);

always @(posedge clk)
begin
        if (msb_check[3])
                pwr_6 <= pwr_5;
        else
                pwr_6 <= { pwr_5[23:0], rng[15:8] };

        log2_6 <= { log2_5[4], msb_check[3], 3'b000 };
```

```
end

// Third stage
assign msb_check[2] = |(pwr_6[31:28]);

always @(posedge clk)
begin
        if (msb_check[2])
                pwr_7 <= pwr_6;
        else
                pwr_7 <= { pwr_6[27:0], rng[7:4] };

        log2_7 <= { log2_6[4:3], msb_check[2], 2'b00 };
end

// Fourth stage
assign msb_check[1] = |(pwr_7[31:30]);

always @(posedge clk)
begin
        if (msb_check[1])
                pwr_8 <= pwr_7;
        else
                pwr_8 <= { pwr_7[29:0], rng[3:2] };

        log2_8 <= { log2_7[4:2], msb_check[1], 1'b0 };
end

// Final stage
assign msb_check[0] =   pwr_8[31];

always @(posedge clk)
begin
        if (msb_check[0])
                pwr_9 <= pwr_8;
        else
                pwr_9 <= { pwr_8[30:0], rng[1] };

        log2_9 <= { log2_8[4:1], msb_check[0] };
        log2_9 <= { log2_8[4:1], msb_check[0] };
end


// ----------
// LUT lookup
// ----------

// Address mapping
assign lut_addr_9 = { 1'b0, pwr_9[30:20], 4'h0 };

// Actual LUT
RAMB36E1 #(
        .RDADDR_COLLISION_HWCONFIG("PERFORMANCE"),
```

```
                .SIM_COLLISION_CHECK("NONE"),
                .DOA_REG(1),
                .DOB_REG(1),
                .EN_ECC_READ("FALSE"),
                .EN_ECC_WRITE("FALSE"),

        .INIT_00(256'h02b202840256022801fa01cd019f01710143011500e700b8008a005c002e00
00),

        .INIT_01(256'h058c055f0531050404d604a9047b044e042003f203c503970369033b030e02
e0),

        .INIT_02(256'h08610834080707da07ad077f0752072506f806ca069d06700642061505e705
ba),

        .INIT_03(256'h0b310b040ad70aaa0a7d0a500a2409f709ca099d09700943091608e908bc08
8e),

        .INIT_04(256'h0dfb0dce0da20d750d490d1c0cef0cc30c960c6a0c3d0c100be40bb70b8a0b
5d),

        .INIT_05(256'h10bf10931067103b100e0fe20fb60f8a0f5d0f310f050ed90eac0e800e530e
27),

        .INIT_06(256'h137e1353132712fb12cf12a31277124b121f11f311c7119b116f1143111710
eb),

        .INIT_07(256'h1639160d15e215b6158a155f1533150814dc14b014851459142d140213d613
aa),

        .INIT_08(256'h18ed18c21897186c1841181517ea17bf17941768173d171216e616bb168f16
64),

        .INIT_09(256'h1b9d1b731b481b1d1af21ac71a9c1a711a461a1b19f019c5199a196f194419
19),

        .INIT_0A(256'h1e481e1e1df31dc91d9e1d741d491d1e1cf41cc91c9e1c731c491c1e1bf31b
c8),

        .INIT_0B(256'h20ee20c4209a20702045201b1ff11fc61f9c1f721f471f1d1ef21ec81e9d1e
73),

        .INIT_0C(256'h23902366233c231222e822be2294226a2240221621ec21c12197216d214321
19),

        .INIT_0D(256'h262c260325d925af2586255c2532250824df24b5248b24612437240d23e423
ba),

        .INIT_0E(256'h28c4289b28712848281e27f527cc27a22779274f272626fc26d326a9267f26
56),

        .INIT_0F(256'h2b572b2e2b052adc2ab32a8a2a612a372a0e29e529bc299229692940291728
ed),
```

```
        .INIT_10(256'h2de62dbd2d942d6b2d432d1a2cf12cc82c9f2c762c4d2c242bfb2bd22ba92b
80),
        .INIT_11(256'h30703047301f2ff62fce2fa52f7d2f542f2b2f032eda2eb12e892e602e372e
0f),
        .INIT_12(256'h32f632cd32a5327d3255322c320431dc31b3318b3162313a311230e930c130
98),
        .INIT_13(256'h3577354f352734ff34d734af3487345f3437340f33e733be3396336e334633
1e),
        .INIT_14(256'h37f437cc37a4377d3755372d370536de36b6368e3666363e361635ef35c735
9f),
        .INIT_15(256'h3a6c3a453a1e39f639cf39a739803958393139093 8e238ba3892386b384338
1b),
        .INIT_16(256'h3ce13cba3c933c6b3c443c1d3bf63bcf3ba73b803b593b313b0a3ae33abb3a
94),
        .INIT_17(256'h3f513f2a3f033edd3eb63e8f3e683e413e1a3df33dcc3da53d7d3d563d2f3d
08),
        .INIT_18(256'h41be41974170414a412340fc40d540af40884061403a40143fed3fc63f9f3f
78),
        .INIT_19(256'h442643ff43d943b3438c4366433f431942f242cc42a5427f42584231420b41
e4),
        .INIT_1A(256'h468a4664463e461745f145cb45a5457f45584532450c44e544bf4499447244
4c),
        .INIT_1B(256'h48ea48c4489e48784853482d480747e147bb4795476f4748472246fc46d646
b0),
        .INIT_1C(256'h4b474b214afb4ad64ab04a8a4a644a3f4a1949f349cd49a84982495c493649
10),
        .INIT_1D(256'h4d9f4d7a4d544d2f4d094ce44cbe4c994c734c4e4c284c034bdd4bb84b924b
6c),
        .INIT_1E(256'h4ff44fcf4faa4f844f5f4f3a4f154eef4eca4ea54e7f4e5a4e354e0f4dea4d
c5),
        .INIT_1F(256'h5245522051fb51d651b1518c51675142511d50f850d350ae50895063503e50
19),
        .INIT_20(256'h5492546e5449542453ff53da53b65391536c5347532252fd52d952b4528f52
6a),
```

```
        .INIT_21(256'h56dc56b75693566e564a5625560155dc55b85593556e554a5525550054dc54
b7),
        .INIT_22(256'h592258fe58d958b55891586c5848582457ff57db57b75792576e5749572557
00),
        .INIT_23(256'h5b645b405b1c5af85ad45ab05a8c5a685a445a2059fb59d759b3598f596a59
46),
        .INIT_24(256'h5da35d805d5c5d385d145cf05ccc5ca85c845c605c3d5c195bf55bd15bad5b
89),
        .INIT_25(256'h5fdf5fbb5f985f745f505f2d5f095ee55ec25e9e5e7a5e565e335e0f5deb5d
c7),
        .INIT_26(256'h621761f461d061ad618961666142611f60fb60d860b46091606d604a602660
03),
        .INIT_27(256'h644c6429640563e263bf639c637863556332630f62eb62c862a56281625e62
3a),
        .INIT_28(256'h667d665a6637661465f165ce65ab658865656542651f64fc64d864b5649264
6f),
        .INIT_29(256'h68ab688868666843682067fd67da67b767946772674f672c670966e666c366
a0),
        .INIT_2A(256'h6ad66ab36a916a6e6a4b6a296a0669e469c1699e697b69596936691368f168
ce),
        .INIT_2B(256'h6cfd6cdb6cb96c966c746c516c2f6c0c6bea6bc76ba56b836b606b3d6b1b6a
f8),
        .INIT_2C(256'h6f226eff6edd6ebb6e996e776e546e326e106dee6dcb6da96d876d646d426d
20),
        .INIT_2D(256'h7143712170ff70dd70bb709970777055703370116fee6fcc6faa6f886f666f
44),
        .INIT_2E(256'h7361733f731d72fb72da72b8729672747252723072 0e71ec71cb71a9718771
65),
        .INIT_2F(256'h757c755a7539751774f574d474b27490746f744d742b740a73e873c673a473
83),
        .INIT_30(256'h779477727751772f770e76ec76cb76aa7688766776457624760275e075bf75
9d),
        .INIT_31(256'h79a879877966794579247902 78e178c0789e787d785c783a781977f877d677
b5),
```

```
        .INIT_32(256'h7bba7b997b787b577b367b157af47ad37ab27a917a707a4e7a2d7a0c79eb79
ca),
        .INIT_33(256'h7dc97da87d887d677d467d257d047ce37cc27ca17c807c5f7c3e7c1d7bfc7b
db),
        .INIT_34(256'h7fd57fb57f947f737f537f327f117ef07ed07eaf7e8e7e6d7e4d7e2c7e0b7d
ea),
        .INIT_35(256'h81de81be819d817d815c813c811b80fb80da80ba809980788058803780177f
f6),
        .INIT_36(256'h83e583c483a4838483638343823830282e282c182a1828182608240821f81
ff),
        .INIT_37(256'h85e885c885a8858885678547852785078 4e784c684a6848684668445842584
05),
        .INIT_38(256'h87e987c987a9878987698749872987 0986e986c986a9868986688648862886
08),
        .INIT_39(256'h89e789c789a7898789678948892889 0888e888c888a8888888688848882988
09),
        .INIT_3A(256'h8be28bc28ba28b838b638b438b248b048ae48ac58aa58a858a668a468a268a
06),
        .INIT_3B(256'h8dda8dbb8d9b8d7c8d5c8d3d8d1d8cfe8cde8cbf8c9f8c808c608c408c218c
01),
        .INIT_3C(256'h8fd08fb18f918f728f538f338f148ef58ed58eb68e978e778e588e388e198d
fa),
        .INIT_3D(256'h91c391a4918591669146912791 0890e990ca90ab908b906c904d902e900e8f
ef),
        .INIT_3E(256'h93b3939493769357933893 1992fa92db92bc929c927d925e923f9220920191
e2),
        .INIT_3F(256'h95a1958395649545952695 0794e894ca94ab948c946d944e942f941093f193
d2),
        .INIT_40(256'h978d976e974f9731971296f396d596b696979679965a963b961c95fe95df95
c0),
        .INIT_41(256'h997599579938991a98fb98dd98be98a0988198639844982698 0797e897ca97
ab),
        .INIT_42(256'h9b5c9b3d9b1f9b019ae29ac49aa69a879a699a4a9a2c9a0e99ef99d199b299
94),
```

```
        .INIT_43(256'h9d3f9d219d039ce59cc79ca99c8a9c6c9c4e9c309c119bf39bd59bb79b989b
7a),
        .INIT_44(256'h9f219f039ee59ec79ea99e8b9e6d9e4f9e309e129df49dd69db89d9a9d7c9d
5e),
        .INIT_45(256'ha100a0e2a0c4a0a6a088a06aa04ca02ea0119ff39fd59fb79f999f7b9f5d9f
3f),
        .INIT_46(256'ha2dca2bea2a1a283a265a247a22aa20ca1eea1d0a1b3a195a177a159a13ba1
1e),
        .INIT_47(256'ha4b6a499a47ba45ea440a422a405a3e7a3c9a3aca38ea371a353a335a318a2
fa),
        .INIT_48(256'ha68ea671a653a636a618a5fba5dda5c0a5a2a585a567a54aa52ca50fa4f1a4
d4),
        .INIT_49(256'ha863a846a829a80ba7eea7d1a7b4a796a779a75ca73ea721a703a6e6a6c9a6
ab),
        .INIT_4A(256'haa36aa19a9fca9dfa9c2a9a5a987a96aa94da930a913a8f5a8d8a8bba89ea8
81),
        .INIT_4B(256'hac07abeaabcdabb0ab93ab76ab59ab3cab1fab02aae5aac8aaabaa8eaa71aa
53),
        .INIT_4C(256'hadd6adb9ad9cad7fad62ad45ad28ad0cacefacd2acb5ac98ac7bac5eac41ac
24),
        .INIT_4D(256'hafa2af85af68af4caf2faf12aef5aed9aebcae9fae82ae66ae49ae2cae0fad
f2),
        .INIT_4E(256'hb16cb14fb133b116b0fab0ddb0c0b0a4b087b06ab04eb031b015aff8afdbaf
bf),
        .INIT_4F(256'hb334b317b2fbb2deb2c2b2a5b289b26cb250b233b217b1fab1deb1c1b1a5b1
88),
        .INIT_50(256'hb4f9b4ddb4c1b4a4b488b46cb44fb433b417b3fab3deb3c2b3a5b389b36cb3
50),
        .INIT_51(256'hb6bdb6a1b684b668b64cb630b614b5f7b5dbb5bfb5a3b587b56ab54eb532b5
15),
        .INIT_52(256'hb87eb862b846b82ab80eb7f2b7d6b7bab79eb781b765b749b72db711b6f5b6
d9),
        .INIT_53(256'hba3dba21ba05b9e9b9ceb9b2b996b97ab95eb942b926b90ab8eeb8d2b8b6b8
9a),
```

```
        .INIT_54(256'hbbfabbdebbc3bba7bb8bbb6fbb54bb38bb1cbb00bae4bac8baadba91ba75ba
59),

        .INIT_55(256'hbdb5bd9abd7ebd62bd47bd2bbd0fbcf4bcd8bcbcbca1bc85bc69bc4dbc32bc
16),

        .INIT_56(256'hbf6ebf53bf37bf1cbf00bee5bec9beadbe92be76be5bbe3fbe24be08bdecbd
d1),

        .INIT_57(256'hc125c10ac0eec0d3c0b7c09cc081c065c04ac02ec013bff7bfdcbfc1bfa5bf
8a),

        .INIT_58(256'hc2dac2bfc2a3c288c26dc251c236c21bc200c1e4c1c9c1aec192c177c15cc1
40),

        .INIT_59(256'hc48dc472c456c43bc420c405c3eac3cfc3b3c398c37dc362c347c32bc310c2
f5),

        .INIT_5A(256'hc63dc622c607c5ecc5d1c5b6c59bc580c565c54ac52fc514c4f9c4dec4c3c4
a8),

        .INIT_5B(256'hc7ecc7d1c7b7c79cc781c766c74bc730c715c6fac6dfc6c4c6a9c68ec673c6
58),

        .INIT_5C(256'hc999c97ec964c949c92ec913c8f9c8dec8c3c8a8c88dc873c858c83dc822c8
07),

        .INIT_5D(256'hcb44cb2acb0fcaf4cadacabfcaa4ca8aca6fca54ca3aca1fca04c9e9c9cfc9
b4),

        .INIT_5E(256'hccedccd3ccb8cc9ecc83cc69cc4ecc34cc19cbfecbe4cbc9cbafcb94cb79cb
5f),

        .INIT_5F(256'hce94ce7ace60ce45ce2bce10cdf6cddccdc1cda7cd8ccd72cd57cd3dcd22cd
08),

        .INIT_60(256'hd03ad01fd005cfebcfd1cfb6cf9ccf82cf67cf4dcf33cf18cefecee4cec9ce
af),

        .INIT_61(256'hd1ddd1c3d1a9d18fd174d15ad140d126d10cd0f1d0d7d0bdd0a3d088d06ed0
54),

        .INIT_62(256'hd37fd365d34bd330d316d2fcd2e2d2c8d2aed294d27ad260d246d22cd211d1
f7),

        .INIT_63(256'hd51ed504d4ead4d1d4b7d49dd483d469d44fd435d41bd401d3e7d3cdd3b3d3
99),

        .INIT_64(256'hd6bcd6a2d689d66fd655d63bd621d607d5eed5d4d5bad5a0d586d56cd552d5
38),
```

```
        .INIT_65(256'hd858d83fd825d80bd7f1d7d8d7bed7a4d78bd771d757d73dd723d70ad6f0d6
d6),

        .INIT_66(256'hd9f3d9d9d9bfd9a6d98cd973d959d93fd926d90cd8f2d8d9d8bfd8a5d88cd8
72),

        .INIT_67(256'hdb8bdb72db58db3fdb25db0cdaf2dad9dabfdaa6da8cda72da59da3fda26da
0c),

        .INIT_68(256'hdd22dd09dcefdcd6dcbcdca3dc8adc70dc57dc3ddc24dc0adbf1dbd8dbbedb
a5),

        .INIT_69(256'hdeb7de9ede84de6bde52de39de1fde06ddedddd3ddbadda1dd87dd6edd55dd
3b),

        .INIT_6A(256'he04ae031e018dfffdfe6dfccdfb3df9adf81df68df4edf35df1cdf03dee9de
d0),

        .INIT_6B(256'he1dce1c3e1aae191e178e15fe145e12ce113e0fae0e1e0c8e0afe096e07de0
63),

        .INIT_6C(256'he36ce353e33ae321e308e2efe2d6e2bde2a4e28be272e259e240e227e20ee1
f5),

        .INIT_6D(256'he4fae4e1e4c8e4afe497e47ee465e44ce433e41ae401e3e8e3cfe3b7e39ee3
85),

        .INIT_6E(256'he686e66ee655e63ce624e60be5f2e5d9e5c0e5a8e58fe576e55de544e52ce5
13),

        .INIT_6F(256'he811e7f9e7e0e7c7e7afe796e77de765e74ce733e71be702e6e9e6d1e6b8e6
9f),

        .INIT_70(256'he99be982e96ae951e938e920e907e8efe8d6e8bee8a5e88ce874e85be843e8
2a),

        .INIT_71(256'heb22eb0aeaf1ead9eac0eaa8ea90ea77ea5fea46ea2eea15e9fde9e4e9cce9
b3),

        .INIT_72(256'heca8ec90ec78ec5fec47ec2eec16ebfeebe5ebcdebb5eb9ceb84eb6beb53eb
3b),

        .INIT_73(256'hee2dee14edfcede4edccedb3ed9bed83ed6bed52ed3aed22ed09ecf1ecd9ec
c1),

        .INIT_74(256'hefafef97ef7fef67ef4fef37ef1fef06eeeeeed6eebeeea6ee8dee75ee5dee
45),

        .INIT_75(256'hf131f119f101f0e8f0d0f0b8f0a0f088f070f058f040f028f010eff8efe0ef
c8),
```

```
        .INIT_76(256'hf2b0f298f280f268f251f239f221f209f1f1f1d9f1c1f1a9f191f179f161f1
49),

        .INIT_77(256'hf42ef417f3fff3e7f3cff3b7f39ff387f370f358f340f328f310f2f8f2e0f2
c8),

        .INIT_78(256'hf5abf593f57bf564f54cf534f51cf505f4edf4d5f4bdf4a5f48ef476f45ef4
46),

        .INIT_79(256'hf726f70ef6f7f6dff6c7f6b0f698f680f669f651f639f622f60af5f2f5daf5
c3),

        .INIT_7A(256'hf8a0f888f870f859f841f82af812f7fbf7e3f7cbf7b4f79cf785f76df755f7
3e),

        .INIT_7B(256'hfa18fa00f9e9f9d1f9baf9a2f98bf973f95cf944f92df915f8fef8e6f8cff8
b7),

        .INIT_7C(256'hfb8efb77fb5ffb48fb31fb19fb02faeafad3fabcfaa4fa8dfa75fa5efa46fa
2f),

        .INIT_7D(256'hfd03fcecfcd5fcbdfca6fc8ffc77fc60fc49fc32fc1afc03fbecfbd4fbbdfb
a5),

        .INIT_7E(256'hfe77fe60fe48fe31fe1afe03fdecfdd4fdbdfda6fd8ffd77fd60fd49fd32fd
1a),

        .INIT_7F(256'hffe9ffd2ffbbffa4ff8dff75ff5eff47ff30ff19ff02feebfed3febcfea5fe
8e),
                .INIT_A(36'h000000000),
                .INIT_B(36'h000000000),
                .INIT_FILE("NONE"),
                .RAM_MODE("TDP"),
                .RAM_EXTENSION_A("NONE"),
                .RAM_EXTENSION_B("NONE"),
                .READ_WIDTH_A(18),
                .READ_WIDTH_B(0),
                .WRITE_WIDTH_A(0),
                .WRITE_WIDTH_B(36),                                     // the RAMB36E1
model fails without this
                .RSTREG_PRIORITY_A("RSTREG"),
                .RSTREG_PRIORITY_B("RSTREG"),
                .SRVAL_A(36'h000000000),
                .SRVAL_B(36'h000000000),
                .SIM_DEVICE("7SERIES"),
                .WRITE_MODE_A("READ_FIRST"),
                .WRITE_MODE_B("READ_FIRST")
        )
        log_lut_I (
                .DOADO(lut_do_11),
                .CASCADEINA(1'b0),
                .CASCADEINB(1'b0),
```

```
                .INJECTDBITERR(1'b0),
                .INJECTSBITERR(1'b0),
                .ADDRARDADDR(lut_addr_9),
                .CLKARDCLK(clk),
                .ENARDEN(1'b1),
                .REGCEAREGCE(1'b1),
                .RSTRAMARSTRAM(rst),
                .RSTREGARSTREG(rst),
                .WEA(4'h0),
                .DIADI(32'h00000000),
                .DIPADIP(4'h0),
                .ADDRBWRADDR(16'h0000),
                .CLKBWRCLK(1'b0),
                .ENBWREN(1'b0),
                .REGCEB(1'b0),
                .RSTRAMB(1'b0),
                .RSTREGB(1'b0),
                .WEBWE(8'h0),
                .DIBDI(32'h00000000),
                .DIPBDIP(4'h0)
        );

        // LSBs mapping
        assign msb_9  = pwr_9[31];
        assign lsbs_9 = pwr_9[19:15];

        // Delay lines to compensate for LUT delay
        delay_bit #(2)    dl_msb  (msb_9,  msb_11,  clk);
        delay_bus #(2, 5) dl_lsbs (lsbs_9, lsbs_11, clk);
        delay_bus #(2, 5) dl_log2 (log2_9, log2_11, clk);


        // -----------
        // Final value
        // -----------

        // Final add & saturation
        always @(posedge clk)
        begin
                if (!msb_11)
                        final_12 <= 16'h0000;
                else
                        final_12 <= { log2_11, lut_do_11[15:0] } + lsbs_11;
        end

        // Mapping
        assign out_12 = final_12[20:5];

endmodule // f15_logpwr
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- use STD.textio.all;                       -- basic I/O
-- use IEEE.std_logic_textio.all;            -- I/O for logic types

entity sqrt is
    Port ( x : in  STD_LOGIC_VECTOR (31 downto 0);
           y : out  STD_LOGIC_VECTOR (31 downto 0));
end sqrt;

architecture Behavioral of sqrt is

    function bit2bit_sq(x: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is
        variable y : STD_LOGIC_VECTOR(2*x'left+1 downto 0);
    -- Returns x^2 by intercalating zeros in the argument,
    -- were x has only one bit different from zero.
    begin
        for i in x'left downto 0 loop
            -- x'right must be zero
            y(2*i):=x(i);
            y(2*i+1):='0';
        end loop;
        return y;
    end;

begin
    process(x)
        variable x_mantissa : STD_LOGIC_VECTOR (22 downto 0);
        variable x_exponent : STD_LOGIC_VECTOR (7 downto 0);
        variable x_sign : STD_LOGIC;
        variable y_mantissa : STD_LOGIC_VECTOR (22 downto 0);
        variable y_exponent : STD_LOGIC_VECTOR (7 downto 0);
        variable y_sign : STD_LOGIC;

        variable ix: STD_LOGIC_VECTOR (25 downto 0);
        variable a : STD_LOGIC_VECTOR (51 downto 0);
```

```
        variable biti : STD_LOGIC_VECTOR (25 downto 0);
        variable r : STD_LOGIC_VECTOR (51 downto 0);
        variable rt : STD_LOGIC_VECTOR (52 downto 0);

        -- variable my_line : line;  -- type 'line' comes from textio
    begin
        x_mantissa := x(22 downto 0);
        x_exponent := x(30 downto 23);
        x_sign := x(31);

        y_sign := '0';

        if (x_exponent="00000000") then -- zero
            y_exponent := (others=>'0');
            y_mantissa := (others=>'0');
        elsif (x_exponent="11111111") then -- infinity
            y_exponent := (others=>'1');
            y_mantissa := (others=>'0');
        else

            if (x_exponent(0)='1') then -- exponent-127 is even
                y_exponent := '0' & x_exponent(7 downto 1) + 64;
                ix := "01" & x_mantissa & '0';
            else -- exponent-127 is odd
                -- shift mantissa one to the left and subtract one from x_exponent
                y_exponent := '0' & x_exponent(7 downto 1) + 63;
                ix := '1' & x_mantissa & "00";
            end if;
            -- mantissa is m=ix/2^24
            -- (one zero was added to the right to make the power even)
            -- let the result of the integer square root algorithm be iy (26 bits)
            -- iy = sqtr(ix)*2^13
            -- resulting that sqrt(m)=iy/2^25

            -- Integer input N bits square root algorithm:
            -- r is be the reminder, r=ix-z^2, and z(N+1) the result,
            --  with bit(N)=1/2^(N/2), and bit(n)=2^(N/2-n)
            -- Test each bit in the result, from the most significative to the least
            -- significative: n goes from zero no N.
            -- if bit is one: r(n+1) = ix - (z(n)+bit(n))^2 =
            --                               r(n) - 2z(n)bit(n) - bit(n)^2
            -- else           r(n+1) = r(n)
            -- bit will be one if the resulting remainder is positive.
            -- making a(n) = 2z(n)bit(n), one has,
            -- if bit is one: a(n+1) = 2(z(n)+bit(n))bit(n)/2 =
            --                                  a(n)/2+bit(n)^2
            -- else           a(n+1) = a(n)/2
            -- and a(N+1) = 2z(N+1)/2^(N/2+1) = z(N+1)/2^(N/2)

            -- VHDL Implementation

            a := (others=>'0');
```

```
            biti := "10" & x"000000"; -- 2^(25)
            -- biti has the bit being evaluated equal to one
            r(51 downto 26):= ix; -- r is in Q26
            r(25 downto 0):=(others=>'0');

            for i in 25 downto 0 loop
                rt := ('0' & r) - ('0' & (a or bit2bit_sq(biti)));
                -- trial for the new value for the reminder
                a := '0' & a(51 downto 1); -- srl
                if (rt(52)='0') then -- rt>=0
                    r := rt(51 downto 0);
                    a := a or bit2bit_sq(biti); -- the adder is safelly replaced by
an or
                end if;
                biti := '0' & biti(25 downto 1); -- srl 1
            end loop;

            a(24 downto 2) := a(24 downto 2)+a(1); -- round
            -- even for ix = all '1' a will not oveflow

            -- a is the result
            y_mantissa := a(24 downto 2);

        end if;

        y(22 downto 0) <= y_mantissa;
        y(30 downto 23) <= y_exponent;
        y(31) <= y_sign;
   end process;
end Behavioral;
```